

Grafische Bedienoberflächen

Lernziele

- ✖ Programmierung einer grafischen Oberfläche zu einem Spiel in **Java mit der GUI-Bibliothek swing** zu einer vorgegeben Spielelogik
- ✖ Reagieren auf Interaktionen
- ✖ Programmierung nach dem **MVC-Modell**
- ✖ Verwendung verschiedener **GUI-Komponenten** und Platzierung mit Hilfe von **Layoutmanagern**
- ✖ Java-Anwendung im Web veröffentlichen
- ✖ **Multimediale Effekte** wie Sound, Video oder Animationen verwenden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

1

Inhalt

- ✖ Fenster und GUI-Komponenten in Java swing
- ✖ GUI-Komponenten platzieren mit Layoutmanagern
- ✖ Reagieren auf Interaktionen
- ✖ Das MVC Programmier-Modell
- ✖ Menüs und Dialoge verwenden
- ✖ Bilder in Java verwenden
- ✖ Multithreading und Threadsicherheit von swing
- ✖ Sound und multimediale Elemente einbinden
- ✖ Eine Java swing Anwendung ins Web stellen
- ✖ Alternative Java GUI-Frameworks (SWT, JavaFX)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

2

Weiterführende Literatur

- ✖ Online Bücher zu Java
 - Christian Ullenboom: [Java ist auch eine Insel](#), Rheinwerk openbook
 - Christian Ullenboom: [Java 7 – Mehr als eine Insel](#), Rheinwerk openbook
- ✖ Bücher zu Java
 - Guido Krüger, Thomas Stark: Handbuch der Java Programmierung, Addison- Wesley
- ✖ Online Tutorials zu Java und swing
 - <http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>
 - <http://docs.oracle.com/javase/tutorial/uiswing/index.html>
 - <http://docs.oracle.com/javase/7/docs/api/>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

3

Die Übungen zur Vorlesung

- ✖ Projektorientiert
 - Es wird die GUI zu einem vorgegeben Spiel **DionaRap** implementiert
 - Die GUI wird in 5 Übungsschritten aufgebaut
 - Die Spielelogik wird zu Verfügung gestellt
- ✖ Eine Musterlösung zu DionaRap findet man in Moodle, Kurs Grafische Bedienoberflächen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

4

Das Spiel DionaRap

- ✖ [Das einfache Original](#)
- ✖ [Die Multithreading Version DionaRap_MT](#)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

5

Die Übungen zur Vorlesung

- ✖ Entwicklungsumgebung ist Eclipse
 - ➔ Open Source und damit für Studenten kostenlos
 - ➔ Kann mit PlugIns erweitert werden z.B. UML-Designer, GUI-Designer, Versionsverwaltung Subversion
 - ➔ Unterstützt sowohl swing als auch SWT und JavaFX

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

6

Zusätzliche Hilfestellungen

- Alle Unterlagen zur Vorlesung wie Folien, Übungsblätter, Termine, Beispielprogramme und Sprechzeiten der Tutoren findet man in der Lernplattform [Moodle](#) im Kurs [Grafische Bedienoberflächen](#)
- Zu den Übungen gibt es in der Lernplattform Moodle ein **Forum** und ein **Wiki**. **Beides sollten Sie intensiv nutzen.**
- Die Bildung von **Lerngruppen** ist sinnvoll
- **Übungsaufgaben** sollten in **Heimarbeit** vorbereitet werden. Die Übungsstunden dienen dazu, die Programme zu verifizieren und unter Hilfestellung zu verbessern. Die Übungsaufgaben dienen der Vorbereitung auf die Klausur.
- **Tutoren** in den Übungsstunden oder Sprechstunden um Hilfe bitten, wenn Sie Fragen zu den Aufgaben, zu den eigenen Lösungen und Problemen mit der Realisierung haben

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

7

Einführung in die GUI-Programmierung

- × Bisher:
 - Eingabe von Tastatur
 - Ausgabe auf Konsole
- × Mit GUI
 - Anwendung ist mit Maus bedienbar
 - Anwendung verwendet mehrere Fenster
 - grafischer Objekte zur Anzeige und Eingabe

10 March 2016

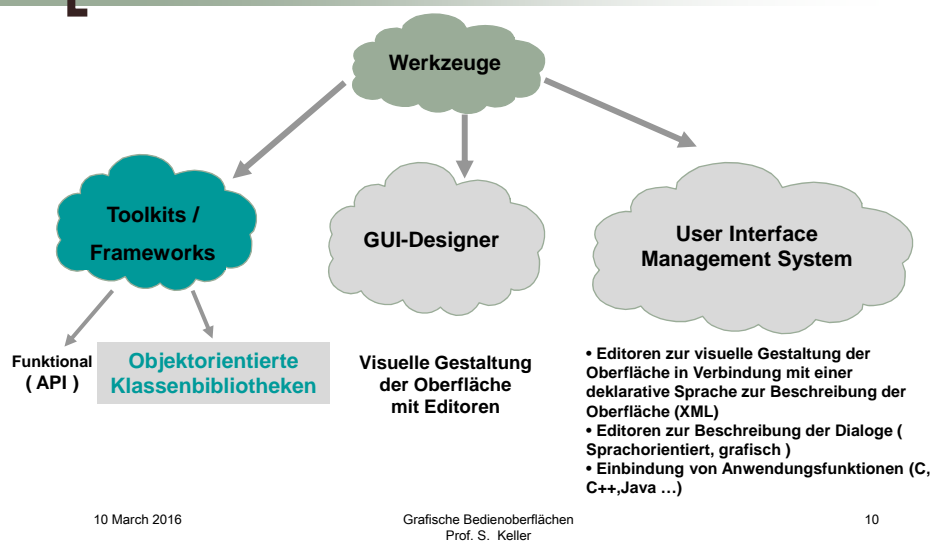
Grafische Bedienoberflächen
Prof. S. Keller

8

GUI von DionaRap



Entwicklungswerkzeuge



Java- Bibliotheken

- ✖ Java Foundation Classes (JFC)
 - beinhaltet AWT und swing
 - Wird in dieser Vorlesung behandelt
- ✖ Standard Widget Toolkit (SWT)
- ✖ JavaFX

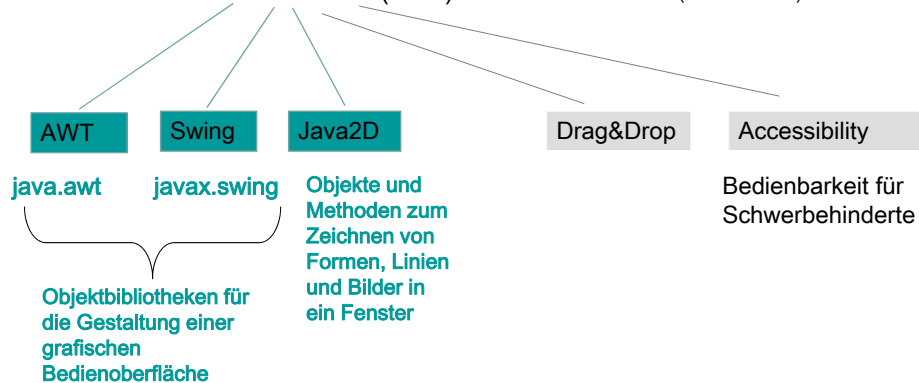
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

11

Grafische Oberflächen in Java Java Foundation Classes (JFC)

Java Foundation Classes (JFC) – ab Java Version 1.2 (aktuell 1.8.40)



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

12

Grafische Oberflächen in Java swing / awt

- ✖ Abstract Windows Toolkit (**AWT**) von SUN
 - Farben, Fonts
 - Eventverarbeitung
 - einfachste Komponenten (veraltet, wurde durch swing ab Java Version 1.3 ersetzt)

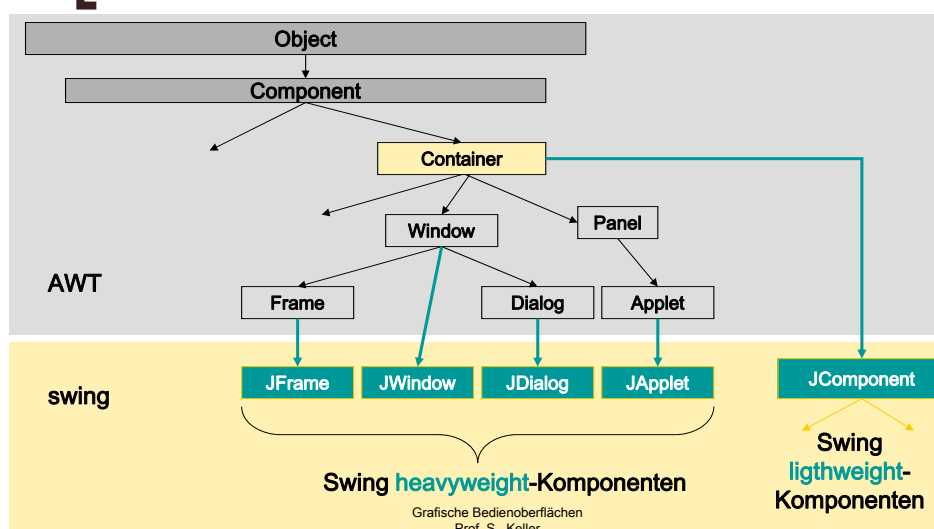
- ✖ **swing** von SUN
 - Widgets zu Gestaltung modernerer Oberflächen
 - am weitesten verbreitet
 - Performance-Probleme
 - stark objektorientiert und einfach zu erlernen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

13

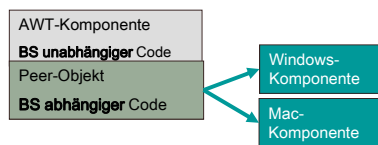
JAVA - Swing Klassenhierarchie



Unterschiede AWT - Swing

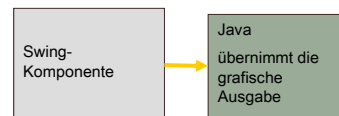
× AWT

- GUI-Komponenten werden auf die Komponenten der GUI des jeweiligen Betriebssystems abgebildet
 - Nur die Schnittmenge aller BS steht zu Verfügung
- Look&Feel immer vom Betriebssystem
- Heavyweight-Komponenten



× swing

- Komponenten werden von Java-Laufzeitumgebung gezeichnet
 - Lightweight-Komponenten
 - Ausnahme:
Fenster werden vom Betriebssystem übernommen



- Look&Feel unabhängig vom Betriebssystem

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

15

Grafische Oberflächen in Java Alternative Technologien / Bibliotheken

× SWT (Standard Widget Toolkit)

- Bibliothek für die Erstellung grafischer Oberflächen mit Java.
- 2001 von IBM für die Entwicklungsumgebung Eclipse entwickelt und wird kontinuierlich gepflegt.
- nutzt dabei im Gegensatz zu Swing die nativen grafischen Elemente des Betriebssystems - wie das AWT von Sun - und ermöglicht somit die Erstellung von Programmen welche eine Optik vergleichbar mit "nativen" Programmen aufweisen.
- Allerdings leidet SWT auf einigen nicht-Windows Plattformen unter Effizienzproblemen, da es viele Features eines Basistoolkits voraussetzt, welche wenn nicht vorhanden emuliert werden müssen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

16

Grafische Oberflächen in Java Alternative Technologien / Bibliotheken

✖ JavaFX

- [Java-Spezifikation](#) von [Oracle](#)
- Soll langfristig swing ablösen
- Die *JavaFX 2.0-Laufzeitumgebung* wird seit der Version [Java SE Runtime 7 Update 6](#) mit installiert

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

17

Entwicklungsumgebung Eclipse

✖ GUI-Builder für Eclipse

„Die drei bekanntesten GUI-Builder sind die kommerziellen Tools [Jigloo](#) und [Window Builder Pro](#) (bestehend aus SWT Designer und Swing Designer) sowie der frei erhältliche [Visual Editor](#).

Seit einiger Zeit gibt es eine Portierung des bekannten NetBeans-GUI Builders für die kommerzielle MyEclipse-IDE. Die Portierung nennt sich [Matisse4MyEclipse](#). Etwas aus der Rahmen dieser Aufstellung fällt [JAXFront](#), das grafische Oberflächen aus einem XML-Schema dynamisch erzeugt.

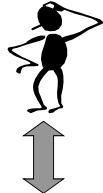
Quelle: <http://wiki.computerwoche.de/doku.php/programmierung/gui-builder-fuer-eclipse>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

18

Architektur interaktiver Systeme



Seeheim-
Schichtenmodell

Seeheim-Konferenz „User Interface
Technik“ 1984, Konstanz

Präsentations- schicht

visuelle, **statische**
Komponenten der
Bedienoberfläche
(Fenster, Tasten,
Grafik, Text....)

Dialog-Management- schicht

Hier werden die **Interaktionen**
vom Benutzer behandelt.
Es wird die Verbindung der
statischen Objekte aus der
Präsentationsschicht mit
Anwendung hergestellt.

Anwendungs- schicht

Hier werden die
Funktionen und
Datenstrukturen
der Anwendung
realisiert.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

19

Einführung

- ✗ GUI wird gebildet aus Fenstern und **grafischen Komponenten** auch **widgets** genannt

→ Zitat:

„**Grafische Benutzeroberfläche** oder auch **grafische Benutzerschnittstelle** (Abk. **GUI** von englisch *graphical user interface*) bezeichnet eine Form von Benutzerschnittstelle eines Computers. Sie hat die Aufgabe, Anwendungssoftware auf einem Rechner mittels grafischer Symbole, Steuerelemente oder auch Widgets genannt, bedienbar zu machen.“

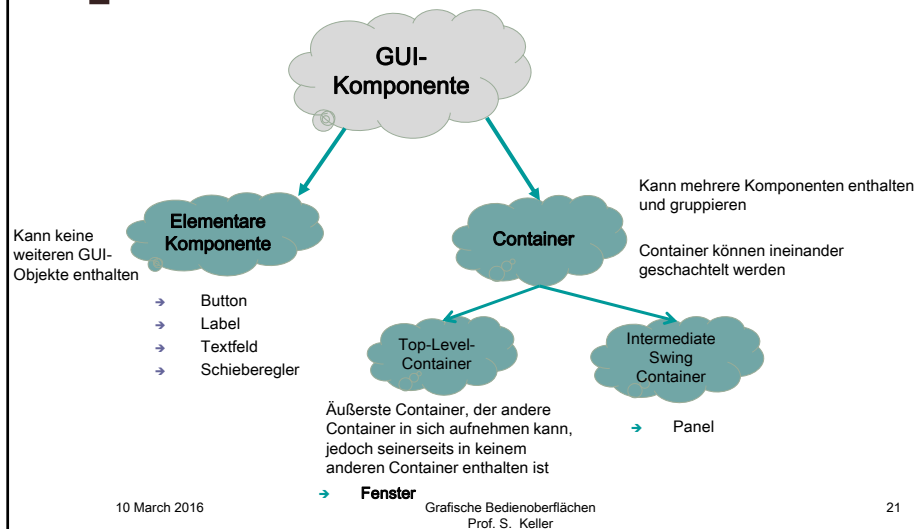
(Quelle: https://de.wikipedia.org/wiki/Grafische_Benutzeroberfläche)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

20

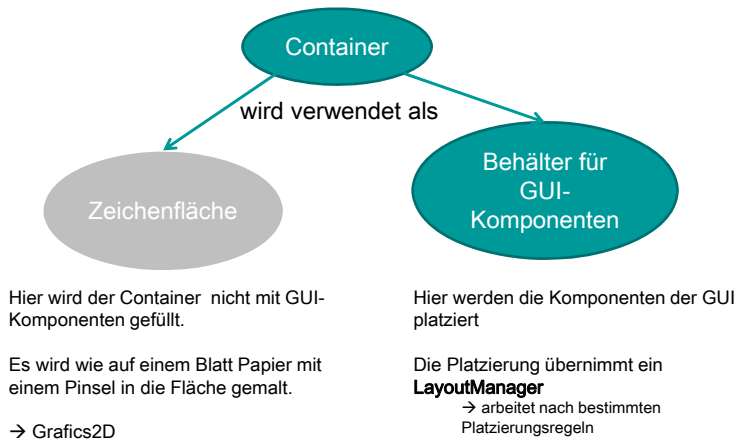
Einführung Klassifikation GUI Komponenten



Klassifikation von Swing-Komponenten

- × **Top-Level-Container**
Sie bilden die oberste Hierarchie einer Swing-Anwendung.
 - JApplet, JDialog, JFrame, JWindow
- × **Universelle Container**
 - JPanel, JScrollBar, JSplitPane, JTabbedPane
- × **Spezielle Container**
In ein Frame können z.B. weitere Fenster gelegt werden und somit die MultiDocumentArchitecture von MFC realisiert werden
 - JInternalFrame, JLayerdPane, JRootPane, JToolBar
- × **Elementare Interaktionen**
 - JButton, JComobox, JList, JMenu, JSlider, JtextField,.....
- × **Nichteditierbare Darstellungsfenster**
 - JLabel, JProgressBar, JToolTip
- × **Editierbare Darstellungsfenster**
 - JTable, JText, JTree, JColorChooser, JFileChooser

Verwendung von Container



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

23

Container

- ✖ Jeder Container besitzt einen Standard Layout-Manager
- ✖ man kann den LayoutManager eines Containers
 - ändern
 - entfernen
 - ohne Verwendung eines LayoutManagers wird die Größe und Position der GUI-Komponenten durch die Angabe von **Koordinaten festgelegt**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

24

Wichtige Methoden von Container

Neue Komponente hinzufügen

`<container>.add(Component)`

Komponente herausnehmen

`<container>.remove(Component)`

Alle Komponenten herausnehmen

`<container>.removeAll()`

Layout festlegen

`<container>.setLayout(LayoutManager)`

Container neu arrangieren

`<container>.validate()`

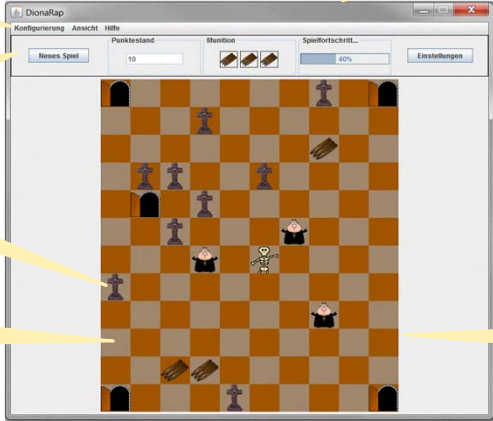
Container neu darstellen

`<container>.repaint()`

Eine Java swing GUI

- ✖ Demo zu den Komponenten in swing
[swingset Demo](#)

Welche Komponenten, Klassen werden für DionaRap benötigt ?



The screenshot shows the DionaRap application window. Callouts point to the following components:

- Anwendungsfenster mit Fensterbalken**: Points to the main application window frame.
- Menüleiste mit Menüs**: Points to the menu bar at the top.
- Leiste für Spielzustand**: Points to the status bar below the menu.
- Bild (Icon)**: Points to a game piece on the board.
- Farbfläche (Label)**: Points to a square on the chessboard.
- Button**: Points to a button in the top right corner.
- Fenster ohne Fensterbalken**: Points to a small window without a title bar.
- Spielbrett (Panel)**: Points to the chessboard area.

10 March 2016 Grafische Bedienoberflächen Prof. S. Keller 27

Welche Komponenten, Klassen werden für Übungsblatt 1 DionaRap benötigt ?

× Fenster

JFrame

JWindow

Anwendungsfenster, in dem das Spielfeld von DionaRap liegt und in dem das Spiel abläuft
Fenster enthält eine Tastatur für die Bedienung des Spielers

× LayoutManager

BorderLayout

GridLayout

Dieses Objekt bestimmt die Anordnung der Komponenten in einem Container
für das Anwendungsfenster
für das Spielfeld und die Tastatur

× Anzeige- / Interaktionselemente

JLabel

JButton

Farbige Spielfelder und Spielfiguren
Taste zum Bewegen des Spielers und zum Schießen

× JPanel

Gruppiert zusammengehörende Interaktionselemente wie das Spielfeld und die Tastatur

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

28

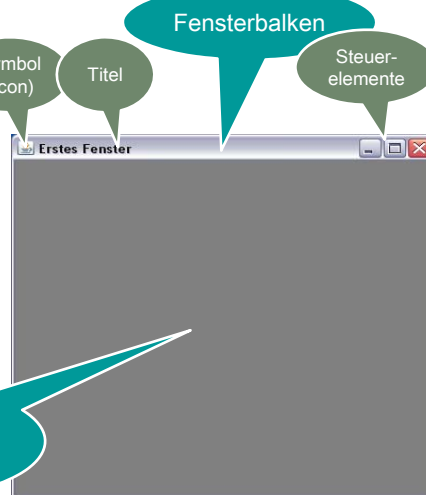
Das Hauptfenster einer Java-Anwendung

- × Eine JAVA-Anwendung mit GUI benötigt für seine Ein- und Ausgaben mindestens ein **JFrame**
- × Ein JFrame besitzt einen Rand und einen **Fensterbalken**.
 - Über den Rand kann man die Fenstergröße verändern
- × In dem Fensterbalken wird ein Icon, ein Titel und die Steuerelemente für das Fenster platziert

Achtung!

Fenster können nur auf dem Bildschirm platziert werden.
In ein Fenster kann kein weiteres Fenster gelegt werden

Darstellungsfläche
Hier werden die GUI-Komponenten platziert



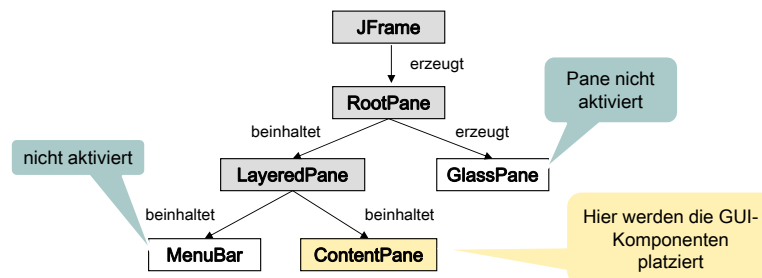
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

29

JFrame Interner Aufbau eines swing-Fensters

- × Ein JFrame setzt sich aus mehreren Subcontainern (**Panes**) zusammen



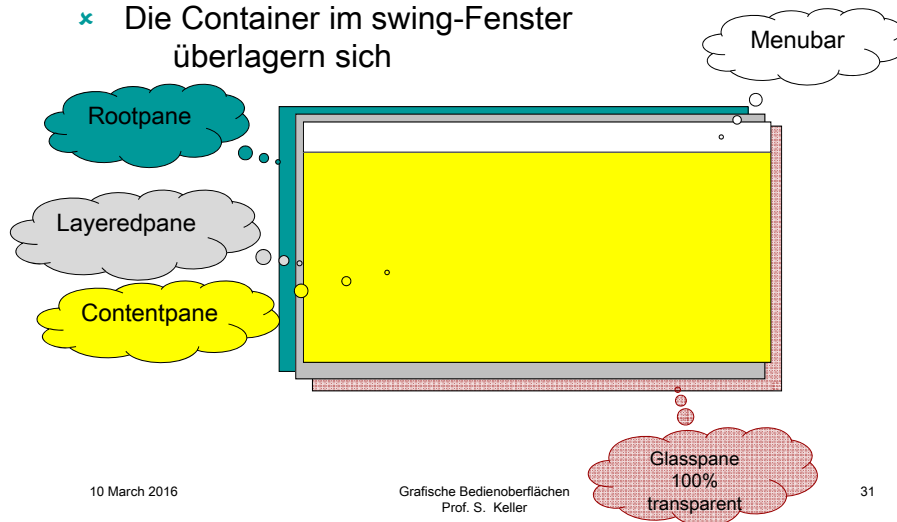
10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

30

JFrame Interner Aufbau eines swing-Fensters

- × Die Container im swing-Fenster überlagern sich



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

31

JFrame Interner Aufbau eines swing-Fensters

- × **Contentpane**
 - Hier werden die GUI-Komponenten eines Fensters platziert
- × **Menubar**
 - Wird nur erzeugt, wenn eine Menüleiste existiert
 - Voreinstellung: nicht vorhanden
- × **Glass Pane**
 - Überdeckt immer alle anderen Container und liegt im Vordergrund
 - 100% transparent
 - erhält alle MouseEvents (falls aktiviert)
 - Voreinstellung: nicht aktiviert

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

32

Das Anwendungsfenster von DionaRap

JFrame

Menüleiste

- × Das Anwendungsfenster eines Java-Programms ist ein **JFrame**

Vorgehensweise:

1. Schritt: Ein Fenster erzeugen und darstellen
2. Schritt: Eigenschaften des Fensters einstellen
3. Schritt: LayoutManager festlegen
4. Schritt: Fenster mit Inhalt füllen
5. Schritt: Fenstergröße dem Inhalt anpassen



Fensterinhalt ist Spielfläche

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

33

JFrame Eigenschaften und Methoden

Eigenschaften

- Aktuelle Größe
- Veränderbarkeit der Fenstergröße
- Titel
- Icon

Methoden zum setzen der Eigenschaft

- `<fenster>.setSize(Breite,Höhe)`
- `<fenster>.setResizable(Boolean)`
- `<fenster>.setTitle(String)`
- `<fenster>.setIconImage(Image image)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

34

JFrame Eigenschaften und Methoden

Eigenschaften

Methoden zum setzen der Eigenschaft

- | | |
|--|--|
| → Aktuelle Position auf dem Bildschirm | <code><fenster>.setLocation(x,y)</code> |
| → Sichtbarkeit | <code><fenster>.setVisible(Boolean)</code> |
| → Hintergrundfarbe | <code><fenster>.setBackground(Color.blue)</code> |

Wichtige JFrame-Methoden

- × `<JFrame>.setVisible(boolean)`
 - Legt fest, ob das Fenster sichtbar oder unsichtbar ist
 - Voreinstellung: `false` (*unsichtbar*)

Achtung!

Ein JFrame wird erst dann sichtbar, wenn die Eigenschaft `visible` auf `true` gestellt wird.

- `setVisible()` stellt das Fenster in der Situation dar, in der es zur Laufzeit aufgerufen wird
 - Sollte erst aufgerufen werden wenn alle Fenstereinstellungen durchgeführt und alle Komponenten im Fenster eingefügt sind

Ein Fenster in swing erzeugen

- ✖ Durch instanziiieren der Klasse JFrame
 - `new JFrame()`
- ✖ Konstruktoren
 - `JFrame()`, `JFrame(String Titel)`

Vorsicht !

Im awt gibt es einige analoge Komponenten.
Nur durch ein vorangestelltes „J“ werden diese unterschieden
z.B. `Frame` / `JFrame` oder `Button` / `JButton`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

37

Erste Schritte zur Implementierung einer Java Anwendung mit GUI

1. Es muß eine **Objektklasse** deklariert werden , die die Methode `main()` enthält.
2. Es muss ein Objekt der Objektklasse **JFrame** instanziiert werden. Nach der Instanziierung ist das JFrame-Objekt zwar erzeugt, aber noch unsichtbar.
3. Das Frame-Objekt muss sichtbar gemacht werden. Dazu dient die Methode `<Frame>.setVisible(true)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

38

Beispiel 1: Anwendung erzeugt eine JFrame-Instanz

```
import javax.swing.JFrame;

public class erstesFenster /* 1. Anwendungsklasse definieren */
{
    public static void main( String arg[] )
    {
        /* 2. Fensterobjekt instanziiieren */
        JFrame anwendungsfenster = new JFrame("DionaRap");

        /* 3. Fenster sichtbar schalten */
        anwendungsfenster.setVisible(true);
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

39

Beispiel 2: Anwendungsklasse wird von JFrame abgeleitet

```
import javax.swing.JFrame;

public class erstesFenster extends JFrame
{
    /* Konstruktor, um die Eigenschaften des Fensters zu setzen */
    erstesFenster(String text)
    {
        super(text);
        this.setVisible(true); /* Fenster anzeigen */
    }

    public static void main( String arg[] ) /* main-Methode */
    {
        /* Hier wird das Fensterobjekt erzeugt */
        new erstesFenster("DioanRap");
    }
}
```

Besser Lösung, da objektorientierter Programmierstil !

Die Anwendungsklasse ist ein Fenster !

Durch Instanziiieren der Anwendungsklasse wird eine Fensterinstanz erzeugt !

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

40

Fenstereigenschaften festlegen

- ✖ Erzeugt man ein swing-Fenster so besitzt diese an der Position (0,0) auf dem Display und besitzt die Fenstergröße 0.
- ✖ Das erzeugte Fenster sieht dann so aus:
 - Screen dump
- ✖ Nach Erzeugen des Fensters (im Konstruktor der Fensterklasse) muss daher zuerst die Position und Größe festgelegt werden, bevor man das Fenster sichtbar macht.

Fenstergröße festlegen

- ✖ Größe fest vorgeben
 - JFrame-Methode `setSize()` verwenden
`<frame>.setSize(Hoehe, Breite)`
- ✖ Größe optimieren auf Fensterinhalt und zur Laufzeit berechnen lassen
 - JFrame-Methode `pack()` verwenden
`<frame>.pack()`

JFrame-Methode pack()

- ✖ Die Größe des Fensters wird auf optimale Breite und Höhe eingestellt, so dass alle im Fenster eingefügten Komponenten optimal dargestellt werden.
 - Berücksichtigt nur die Komponenten, die zum Zeitpunkt des Aufrufs von pack() im Fenster liegen
 - Ruft von jeder Komponente im Fenster die Methode `getPreferredSize()` auf

Achtung !

Sind im Fenster noch keine GUI-Elemente platziert, wird nur der Fensterbalken dargestellt.

Beispiel 4: Fenstergröße festlegen

```
import javax.swing.JFrame;

public class erstesFenster extends JFrame
{
    erstesFenster()
    {
        /* Anwendung soll bei Exit beendet werden */
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(200,200); /* Fenstergröße fest einstellen */
        this.setVisible(true); /* Fenster anzeigen */
    }

    public static void main( String arg[] )
    {
        /* Hier wird das Fensterobjekt erzeugt */
        new erstesFenster();
    }
}
```

Achtung !

Im Fenster sind noch keine Komponenten platziert, daher ist pack() nicht sinnvoll

Position des Fensters festlegen

- ✖ Absolute Position über x,y-Koordinate auf dem Display angeben
 - `<fenster>.setLocation(x,y)`
- ✖ Fenster mittig auf dem Bildschirm platzieren
 - `<fenster>.setLocationRelativeTo(null)`
 - Dies funktioniert allerdings erst, wenn die Größe des Fensters mit `setSize()` oder `pack()` gesetzt wurde

Beenden einer Anwendung in swing

- ✖ In einem swing-JFrame muss die Reaktion auf das Drücken des Exit-Knopfes mit der Methode

`<JFrame>.setDefaultCloseOperation()`

eingestellt werden !

Vorsicht !

Fehlt diese Anweisung, so verschwindet das Fenster vom Bildschirm, die Anwendung läuft aber im Hintergrund ohne Fenster weiter.

JFrame Methode setDefaultCloseOperation(int operation)

- ✖ Mit dieser Methode kann man angeben, wie ein Fenster auf das drücken des Exit-Knopfes reagieren soll.
 - Mögliche Werte für den Parameter operation sind:
 - **DO_NOTHING_ON_CLOSE** (definiert in WindowConstants)
Tue nichts.
 - **HIDE_ON_CLOSE** (definiert in WindowConstants)
Der Frame wird unsichtbar
 - **DISPOSE_ON_CLOSE** (definiert in WindowConstants):
Lösche das Fensterobjekt
 - **EXIT_ON_CLOSE** (definiert in JFrame)
Verlasse die Anwendung mit Aufruf der System-Methode exit(0)

Als Voreinstellung wird die Operation **HIDE_ON_CLOSE** eingestellt

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

47

Beispiel 3: Anwendung korrekt beenden

```
import javax.swing.JFrame;

public class erstesFenster extends JFrame
{
    erstesFenster()
    {
        /* Anwendung soll bei Exit beendet werden */
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);           /* Fenster anzeigen */
    }

    public static void main( String arg[] )
    {
        /* Hier wird das Fensterobjekt erzeugt */
        new erstesFenster();
    }
}
```

Achtung !

Das Fenster ist zwar sichtbar aber die Fenstergröße ist nur minimal. Es wird nur der Fensterbalken angezeigt.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

48

Komponenten in einem Fenster platzieren

- ✖ Wie und Wo werden GUI-Komponenten in einen Fenster platziert?

GUI-Komponenten

swing

- ✖ Basisklasse für alle swing- **GUI-Komponenten** ist die Klasse **JComponent**
- ✖ JComponent vererbt folgende Eigenschaften an jede GUI-Komponente
 - **Pluggable Look&Feel**
 - Da die GUI-Komponenten unabhängig vom Betriebssystem durch Java gezeichnet werden, kann man zur Laufzeit kann das Erscheinungsbild der GUI ändern
 - **ToolTip**
 - Methode: `setToolTip()`
 - Jede Komponente kann ohne Maus über die Tastatur ausgewählt werden
 - Methode: `setMnemonic()`
 - Minimale, maximale und bevorzugte Größe
 - `setPreferredSize()`, `setMinimalSize()`, `setMaximalSize()`
 - Ausrichtung in einem Container
 - `setAlignmentX()`, `setAlignmentY()`

Plugable Look-And-Feel (LAF)

- ✖ In swing kann zur Laufzeit das Look&Feel von Komponenten der GUI verändert werden
 - Standard: **Metal** (Java eigene grafische Erscheinung)
 - Weitere vorinstallierte Look&Feels: **Windows**, **Unix-Motiv**
 - Es können auch eigene erstellte Look&Feels installiert werden
 - Im Internet findet man viele installierbare LAF's (theme pack)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

51

ToolTips/ Mnemonic

- ✖ Was ist ein Tooltip ?
 - Ein Tooltip ist ein kurzer erklärender Text. Wenn man mit der Maus über die GUI-Komponente fährt, erscheint ein kleines Textfeld mit dem erklärenden Text
- ✖ Was heißt Mnemonic ?
 - Komponenten können über Tastenkombination ausgewählt werden

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

52

Beispiel ToolTips/ Mnemonic

```
public class JComponentExmpl extends JFrame
{
    JButton button1, button2;

    public JComponentExmpl()
    {
        button1= new JButton("Button1");
        button2= new JButton("Button2");
        button1.setToolTipText("Das ist ein Button");
        button2.setMnemonic('B');    // button2 anzusprechen mit Alt + B
        this.getContentPane().add(button1, BorderLayout.EAST);
        this.getContentPane().add(button2, BorderLayout.WEST);
    }

    public static void main(String[] args){...};
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

53

Wichtige Eigenschaften von swing Komponenten

Eigenschaft

set- / get-Methoden der Klasse

Hintergrundfarbe
Vordergrundfarbe
Schriftart
Aktuelle Größe

setBackground(Color), get Background()
setForeground(Color), getForeground()
setFont(Font), getFont()
setSize(int Breite, int Höhe),
setSize(Dimension), getSize(), getHeight(),
getWidth(),
set/getPreferredSize(Dimension),
set/getMinimumSize(Dimension),
set/getMaximumSize(Dimension)

Minimale-/maximale und bevorzugte
Größe (nur swing)

Position

setLocation(int x, int y), setLocation(Point),
getLocation().getX(), getY()

sichtbar / unsichtbar
durchsichtig / undurchsichtig

setVisible(boolean), isVisible()
setOpaque(), isOpaque()

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

54

Eigenschaft unsichtbar vs. durchsichtig

- × Eigenschaft **unsichtbar**
 - die Komponente wird nicht dargestellt.
 - man kann Sie daher auch nicht mit der Maus anwählen.
 - Wird von `pack()` nicht berücksichtigt
- × Eigenschaft **durchsichtig**
 - Die Komponente wird dargestellt, ist damit sichtbar
 - Hintergrund ist durchsichtig
 - Vordergrund ist sichtbar
 - Beispiel **JLabel**: Schrift auf durchsichtigem Hintergrund

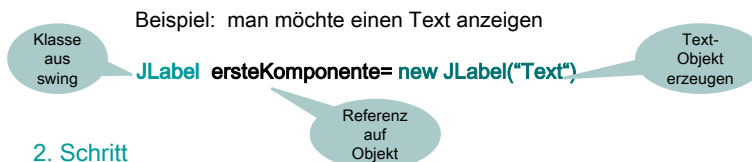
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

55

Einfügen von GUI-Komponenten in ein Fenster

1. Schritt: Erzeugen der GUI-Komponente durch Instanzieren einer Objektklasse aus swing



2. Schritt

Das erzeugte Objekt muss in das Fenster eingefügt werden. Dazu wird die Methode `add()` verwendet.

```
<fenster>.add(ersteKomponente);
```

Die Größe und Anordnung der GUI-Komponenten im Fenster erfolgt nach bestimmten Layoutregeln, die durch den gewählten **LayoutManager** festgelegt werden.

10 March 2016

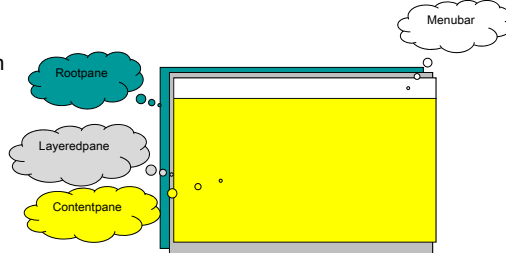
Grafische Bedienoberflächen
Prof. S. Keller

56

GUI-Komponenten einfügen in ein Fenster

- ✖ Wo werden in einem JFrame swing-Komponenten eingefügt und dargestellt ?

- GUI-Komponenten werden im **ContentPane** des Fensters eingefügt und dargestellt
- Menüs werden in der **MenuBar** dargestellt



Beispiel:

```
meinFrame.getContentPane().add(new JButton("OK"));
```

Achtung !

Ab der Version Java 1.5 kann man den Aufruf der Methode getContentPane() weglassen. Java sorgt dafür, dass mit add() die Komponenten im ContentPane platziert werden.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

57

Beispiel Eine Taste im Fenster platzieren

```
import javax.swing.*;
import java.awt.*;          /* wird fuer Color und Container benoetigt */

public class BeispielSwingFrame extends JFrame
{
    public BeispielSwingFrame()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400,400);
        this.setTitle("Dezimal nach Dual Umrechnung");

        /* ContentPane für die Komponenten des Fensters besorgen*/
        Container darstellungsflaeche=this.getContentPane();

        /* Hintergrundfarbe des ContentPane setzen */
        darstellungsflaeche.setBackground(Color.blue);

        /* Taste ins Fenster (in den ContentPane) legen */
        meineTaste = new JButton("OK");
        darstellungsflaeche.add(meineTaste);

        this.setVisible(true);
    }
    public static void main( String arg[] )
    {
        new BeispielSwingFrame();
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

58

Anordnung von GUI-Komponenten im Container

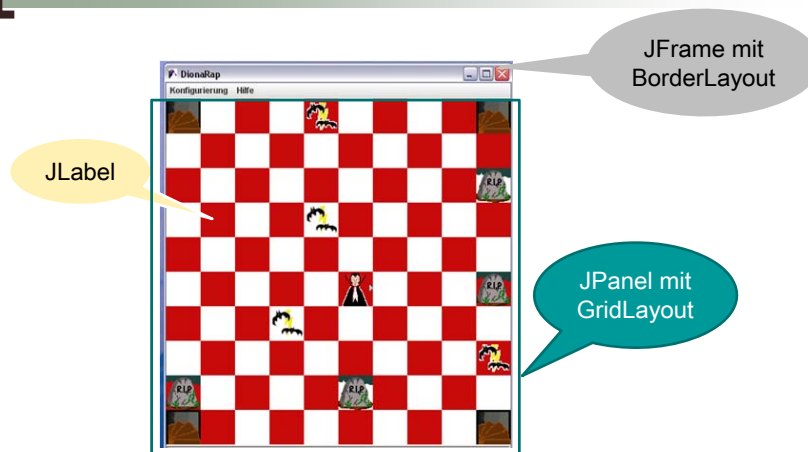
- ✖ Größe und Position von GUI-Elementen im Contentpane des Fensters wird durch ein **LayoutManager** festgelegt
- ✖ Zu jedem Container gehört ein voreingestellter LayoutManager
 - der **LayoutManager** eines JFrames ist vom Typ **BorderLayout**
 - der **LayoutManager** eines JPanels ist vom Typ **FlowLayout**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

59

Verwenden von LayoutManagern in DionaRap



10 March 2016

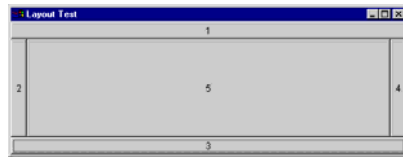
Grafische Bedienoberflächen
Prof. S. Keller

60

BorderLayout

awt

- × **StandardLayout von Fenstern**
 - JFrame, JWindow, JDialog
- × Das Fenster wird in **5 Bereiche** eingeteilt.
 - Oben, Unten, Links, Rechts und Mitte



- × Klassenkonstanten für die Angabe der Gebiete
 - **NORTH, SOUTH, EAST, WEST, CENTER**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

61

BorderLayout

awt

- **Abhängig vom verwendeten Layoutmanager kann /muss beim Einfügen einer Komponente in das Fenster zusätzliche Angaben gemacht werden**

BorderLayout

Wird eine GUI-Komponente in ein BorderLayout gelegt, muss beim **add-Aufruf das Gebiet** angegeben werden, in das die Komponente gelegt werden soll. **Ohne Angabe wird die Komponente in der Mitte platziert**

Beispiel:

```
<container>.add(Component, BorderLayout.NORTH);
```

```
<container>.add(Component);
```

Komponente wird oben im Bereich NORTH platziert

Komponente wird in der Mitte im Bereich CENTER platziert

10 March 2016

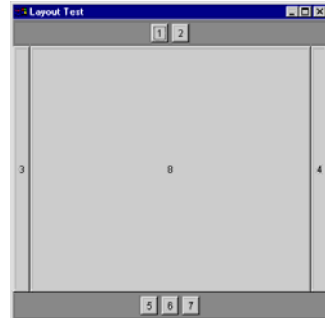
Grafische Bedienoberflächen
Prof. S. Keller

62

BorderLayout

- × Es wird in jedem Bereich genau eine Komponente platziert

- will man z.B. mehrere Tasten im oberen und unteren Bereich platzieren, muss man zwei Panels erzeugen, die Tasten in die Panels legen und das Panel dem Bereich zuordnen



BorderLayout

awt

- Auch die Größe der GUI-Komponente wird durch den LayoutManager festgelegt
 - Die Komponente in der Mitte wird der Größe des Bereichs Mitte (der Fenstergröße) angepasst
 - Die Größe von Mitte berechnet sich aus der Fenstergröße und den darumliegenden Bereichen
 - Für die Bereiche Oben, Unten, Links und Rechts wird die Fenstergröße und die Komponentengröße verwendet
 - Leere Bereiche werden nicht dargestellt

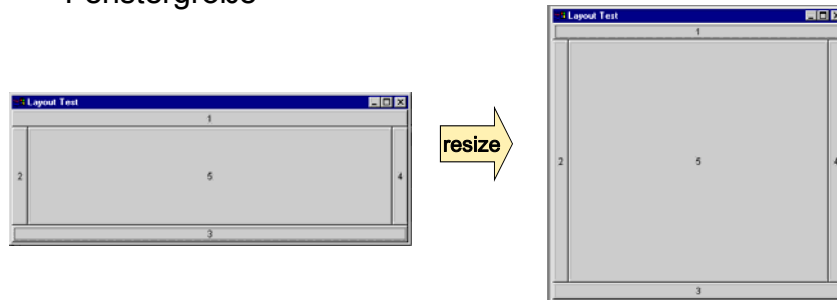
Achtung !

Alle LayoutManager des awt berücksichtigen bei der Größenberechnung nicht die swing-Eigenschaften wie minimale, maximale und bevorzugte Größe. Auch die Ausrichtung einer Komponente wird nicht berücksichtigt.

BorderLayout

awt

- ✖ Darstellung der Komponenten bei Änderung der Fenstergröße



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

65

BorderLayout

- ✖ Abstände zwischen den Bereichen
 - Normalerweise gibt es im BorderLayout zwischen den Bereichen keine Abstände
 - Möchte man zwischen den Bereichen Abstände haben, muss man den voreingestellten LayoutManager löschen und dem Fenster ein neues Layoutobjekt zuordnen.
 - Das neue BorderLayout-Objekt muss dann mit dem Konstruktor:

```
BorderLayout( int horizontalerAbstand, int vertikalerAbstand )
```

erzeugt werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

66

Beispiel – BorderLayout

```
...
public class BeispielBorderLayout extends JFrame
{
    public BeispielBorderLayout()
    {
        // JFrame besitzt BorderLayout
        // Die Buttons werden daher mit Hilfe des BorderLayouts angeordnet
        Container fenster = this.getContentPane();
        fenster.add(new Button("Button1"), BorderLayout.NORTH);
        fenster.add(new Button("Button2"), BorderLayout.SOUTH);
        fenster.add(new Button("Button3"), BorderLayout.WEST);
        fenster.add(new Button("Button4"), BorderLayout.EAST);
        fenster.add(new Button("Button5"), BorderLayout.CENTER);

        this.setSize(300,200);
        this.setVisible(true);
    }
    public static void main(String[] args){ new BeispielBorderLayout(); }
};
```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

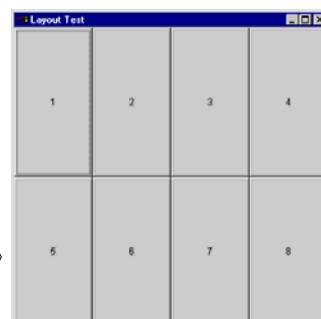
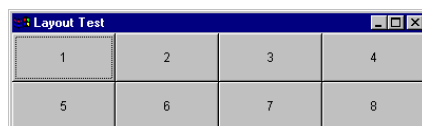
67

GridLayout

awt

- ✗ der Container wird in ein **Gitternetz** mit n Zeilen und m Spalten eingeteilt

- Die Zellen im Grid habe alle **gleiche Grösse**
- Die Komponenten werden von **links nach rechts, zeilenweise von oben nach unten** in die Gitterfelder platziert.
- Die Komponenten werden in eine Zelle eingepasst, d.h. die **Komponentengrösse wird der Gittergrösse angepasst**.



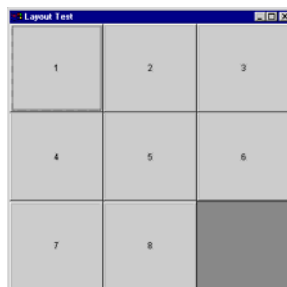
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

68

GridLayout

- **Je Zelle wird eine Komponenten platziert.**
 - Daher sollten mindestens soviel Gitterfelder wie Komponenten vorhanden sind erzeugt werden



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

69

GridLayout

✖ Konstruktoren

- GridLayout()
- GridLayout(int AnzahlZeile, int AnzahlSpalte)
- GridLayout(int AnzahlZeile,int AnzahlSpalte,int hAbstad,int vAbstand)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

70

Komponenten in ein GridLayout legen

1. Schritt: Erzeugen eines LayoutManager-Objektes, hier GridLayout

```
/* Grid mit 4 Zeilen, 3 Spalten und jeweils 5 Pixel Abstand  
zwischen den Gridelementen erzeugen */  
GridLayout meinGrid = new GridLayout(4,3,5,5);
```

2. Schritt: Mit der Methode `setLayout(LayoutManager)` dem Container das GridLayout zuordnen

```
meinFenster.setLayout(meinGrid);
```

Achtung!

Mit `<Container>.setLayout(null)` wird eine vorhandener LayoutManager entfernt.
GUI-Komponenten müssen im Container jetzt im Koordinatensystem positioniert werden.

3. Schritt: Tasten `nacheinander` in ein Grid legen

```
for ( int i=0; i<11; i++ )  
{ meinFenster.add(new JButton(Integer.toString(i)); }
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

71

Komponenten in ein GridLayout legen

- × Tasten in ein **bestimmtes Gridelement** legen

- Man kann in der add-Methode eine Position angeben
- Die angegebene Position, darf nur eine Position sein, die eine Taste an ein gefülltes Grid anfügt oder in der schon eine Taste liegt.
 - Im letzten Fall wird die Taste, die an der Grid-Position liegt und alle folgenden Tasten im Grid nach rechts verschoben
 - Liegt vor der angegebenen Position eine leere Gitterzelle wirft die Methode eine Exception
- **Beispiel: Tastatur.java**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

72

Das Spielfeld realisieren

Klassen JLabel und JPanel

- ✖ Ein JLabel wird verwendet um **Beschriftungen** oder **Symbole** auf der Oberfläche zu platzieren
- ✖ Ein JLabel ist ein Anzeigeobjekt und nicht für die Auswahl mit der Maus gedacht
- ✖ Text oder Bild wird im Normalfall auf einem durchsichtigen Hintergrund und ohne Rand dargestellt

JPanel gruppiert die Labels zu einem Spielbrett



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

73

Klasse JPanel

- ✖ Ein JPanel ist ein Container, in dem mehrere GUI-Elemente zu einer Gruppe zusammengefasst werden können
- ✖ ein rechteckiges Fenster ohne Rahmen
- ✖ arbeitet mit dem Layoutmanager **FlowLayout**

10 March 2016

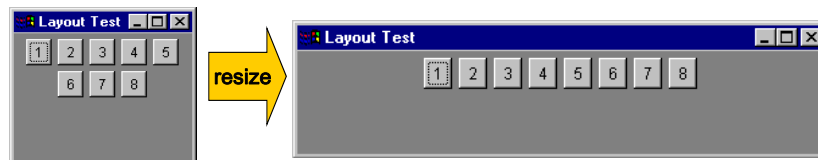
Grafische Bedienoberflächen
Prof. S. Keller

74

FlowLayout

awt

- ✖ Standardlayout von **JPanel**
- ✖ Komponenten werden der Reihe nach von **links nach rechts mit Umbruch** am Containerrand angeordnet
 - Die Komponenten behalten ihre **natürliche Größe**. Werden daher beim Verkleinern oder Vergrößern des Containers in der Größe nicht angepasst.



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

75

FlowLayout

✖ Konstruktoren

- `FlowLayout()`
- `FlowLayout(int Ausrichtung)`
- `FlowLayout(int Ausrichtung, int hAbstand, int vAbstand)`
- Die Ausrichtung kann einer der drei Klassenkonstanten sein
 - `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

76

Beispiel – FlowLayout

```
...
public class BeispielFlowLayout extends JFrame{

    public BeispielFlowLayout(){
        Container fenster = this.getContentPane();
        fenster.setLayout(new FlowLayout(FlowLayout.LEFT,20,20));

        // 5 Buttons werden erzeugt und mit Hilfe des FlowLayouts angeordnet
        fenster.add(new Button("Button 1"));
        fenster.add(new Button("Button 2"));
        fenster.add(new Button("Button 3"));
        fenster.add(new Button("Button 4"));
        fenster.add(new Button("Button 5"));

        this.pack();
        this.setVisible(true);
    }
    public static void main(String[] args){ new BeispielFlowLayout(); }
};
```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

77

Klasse JLabel

- ✖ Da ein Label standardmäßig durchsichtig ist, kann man ein leeres, eingefärbtes Label, wie wir es für ein Spielfeld benötigen, nur dann sehen, wenn man das Label undurchsichtig setzt
- ✖ Über Methoden der Klasse kann man das Label mit einer Hintergrundfarbe versehen, undurchsichtig machen und ihm auch einen Rand geben.
 - Label deckend darstellen
`label.setOpaque(true)`
 - Hintergrundfarbe setzen
`label.setBackground(new Color(155,0,0));`
 - Rand des Labels darstellen
`label.setBorder(BorderFactory.createEtchedBorder());`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

78

Farben verwenden

- × Durch die Objektklasse **Color** werden Farben repräsentiert
 - Intern speichert JAVA die drei Farbwerte in einem int-Wert
 - Bit 0-7 Blau, Bit 8-15 Grün, Bit 16-23 Rot, Bit 24-31 Alpha
 - Ohne Angabe von Alpha wird als Alpha-Wert immer 255 genommen, d.h. die Farbe ist immer voll deckend.
- × Erzeugen einer Farbe durch
 - `new Color(int Rot, int Grün, int Blau)`
 - Die Farbe wird mit dem Alphawert 255, also nicht transparent erzeugt.
 - `new Color(int rot, int grün, int blau, int transparenz)`
- × Ermitteln der Farbe durch
 - `getColor(), getRed(), getGreen(), getBlue()`
- × Globale Konstanten der Klasse Color
 - white, black, gray, yellow, blue, red, green, pink, magenta,

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

79

Klasse JLabel

Package javax.swing

- × Erzeugen eines Labels

`new JLabel()`
`new JLabel(String text)`

Leeres Label anzeigen
Label mit Text anzeigen

`new JLabel(String text, int horizontalAlignment)`

Text des Labels
ausrichten

- Für die Ausrichtung sind folgende Klassenkonstanten möglich:
LEFT, CENTER, RIGHT

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

80

Beispiel JLabel

```
public class labelbeispiel extends JFrame
{
    public labelbeispiel()
    { this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        GridLayout meinlayout=new GridLayout(2,4);
        this.setLayout(meinlayout);

        JLabel label1=new JLabel("Label 1 links deckend in rot ohne Rand",JLabel.LEFT);
        label1.setBackground(Color.red);
        label1.setOpaque(true);

        JLabel label2=new JLabel("Label 2 rechts durchsichtig ohne Rand",JLabel.RIGHT);

        JLabel label3=new JLabel("Label 3 Mitte deckend mit Rand",JLabel.CENTER);
        label3.setBorder(BorderFactory.createEtchedBorder());
        label3.setOpaque(true);
        label3.setBackground(new Color(150,150,150)); ...
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

81

Das Navigationsfenster von DionaRap

- × Das Navigationsfenster ist von der Klasse **JWindow** abgeleitet

JButton



JWindow

- × JWindow ist ein Fenster
 - ohne Fensterbalken
 - ohne Rand

Achtung !

Es gibt keine Möglichkeit dem JWindow einen Rand oder einen Fensterbalken zu geben

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

82

Klasse JWindow

swing

- × StandardLayout ist [BorderLayout](#)
- × Das Fenster kann nicht innerhalb eines Vaterfensters platziert werden, sondern liegt immer auf dem Desktop

Hinweis!

Möchte man ein Fenster innerhalb eines Fensters platzieren muss man die swing-Komponente [JInternalFrame](#) verwenden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

83

Fenster einen Rand geben

- × Um dem Fenster einen Rand zu geben, kann man folgenden Trick anwenden:

Man legt in das Fenster ein JPanel.

Das Panel kann man mit einem Rand ausstatten. Man verwendet dazu eine statische Methode der Klasse [BorderFactory](#). Diese Klasse stellt verschiedene Ränder zu Verfügung.

Beispielsweise kann man dem Panel einen Rand mit Titelbalken geben durch die Verwendung der Methode:

```
panel.setBorder(BorderFactory.createTitledBorder("Titel"));
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

84

Klasse BorderFactory

- ✖ Diese Klasse stellt mehrere statische Methoden zu Verfügung, um Objekten einen Rand zu geben

Beispiele:

- Linie als Rand
 - `BorderFactory.createLineBorder(Color Farbe)`
 - `BorderFactory.createLineBorder(Color Farbe, int Liniendicke)`
- Linie mit Titelleiste
 - `BorderFactory.createTitledBorder(String Titel)`

Weitere Methode können Sie in der Doku zu swing auf den Webseiten von Sun finden:

<http://java.sun.com/javase/6/docs/api/>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

85

Klasse JWindow

- ✖ Ein JWindow sollte immer als **Kindfenster** erzeugt werden
 - nicht den Standardkonstruktor verwendet, sondern einen Konstruktor , dem man das Vaterfenster übergeben kann
 - Wenn man das JWindow als Kindfenster zum JFrame erzeugt, so wird beim Iconify, Delconify und Schliessen des Hauptfensters das Kindfenster mit dem Hauptfenster geöffnet und geschlossen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

86

Beispiel JWindow als Kindfenster verwenden

```
public class EinfachesFenster extends JWindow
{
    EinfachesFenster(JFrame fenster)
    {
        super(fenster);
        this.setSize(100,100); // Hier die Komponenten ins
                               Fenster einfügen
        this.setVisible(true); } }

```

JWindow Konstruktoren

JWindow()

→ Sollte nicht verwendet werden

JWindow(JFrame vaterfenster)

JWindow(JWindow vaterfenster)

Tasten für die Bewegung des Spielers

Klasse JButton

- ✖ Ein JButton ist ein Interaktionsobjekt
 - mit der Maus auswählbar
- ✖ dient zur Auswahl von bestimmten Aktionen
 - mit dem Button ist ein `ActionCommand` verbunden
 - `ActionCommand` ist ein String
- ✖ Ein Button kann mit einem Text beschriftet werden
 - Methode: `<button>.setText(String text)`
 - Die Beschriftung wird als „`ActionCommand`“ interpretiert
- ✖ Wie bei jeder Komponente kann die Hintergrundfarbe, der Font, die Farbe der Beschriftung, die Größe u.v.m. festgelegt werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

89

Klasse JButton ActionCommand

- ✖ Festlegen eines ActionCommand
 - Besitzt die Taste keinen Text, kann das „`ActionCommand`“ mit der Methode
 - `<button>.setActionCommand(String Command)`explizit festgelegt werden.
 - Über diese Methode kann das `ActionCommand` unterschiedlich zur Beschriftung festgelegt werden
- ✖ ActionCommand abfragen
 - `String Button.getActionCommand()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

90

Klasse JButton

✖ Konstruktoren

JButton()

Taste ohne Textinhalt anzeigen

JButton(String text)

Taste mit Text anzeigen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

91

Schriftarten

awt

- ✖ Durch die Objektklasse **Font** werden Schriftarten repräsentiert
- ✖ Erzeugen einer Schriftart
 - ➔ **new Font(String Fontname, int Schriftstil, int Schriftgroesse)**
 - Fontnamen wie „Serif“, „SansSerif“, „Monospaced“ sind in jedem JAVA-System möglich. Serif und SansSerif sind Proportionalschriften.
 - Die Schriftgröße wird in Punkt angegeben
- ✖ **Globale Konstanten** in der Klasse Font
 - ➔ PLAIN, ITALIC, BOLD, ITALIC+BOLD

Beispiel:

```
/* Font für Schriften festlegen */  
Font f = new Font("Arial", Font.BOLD, 14);  
<component>.setFont(f);
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

92

Java BestPractice

- ✖ Kurze Wiederholung wichtiger Programmierkonzepte aus Java
 - Objekt Konstanten
 - Klassenkonstanten und Klassenmethoden
 - Gültigkeitsbereiche private, protected, public
 - Verwendung von Interfaces
 - Beispiele aus der Spielelogik

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

93

Ereignisse in JAVA

- ✖ Es gibt unterschiedliche Ursachen für ein Ereignis
 - Tasten der Tastatur werden gedrückt
 - Mausbewegungen, Mausaktionen
 - Benutzeraktionen
 - der Benutzer wählt ein bestimmtes grafisches Objekt aus
 - Eingabe von Text in ein Eingabefeld
 - Öffnen und Schließen von Fenstern
 - weitere Ursachen

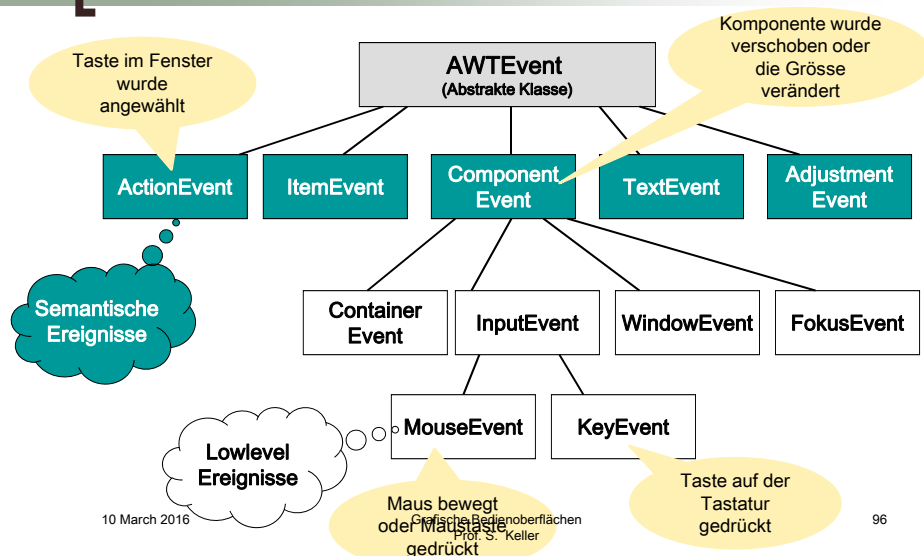
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

94

- ✖ Für jedes Ereignis wird in Java ein Event erzeugt
- ✖ Events werden im awt behandelt

Mögliche Ereignisse (Events) in JAVA



Events

- × Events haben generelle Eigenschaften
 - Zeitpunkt, wann der Event ausgelöst wurde
 - Objekt (*Source*), welches den Event erzeugt hat
- × zusätzlich weitere typabhängige Eigenschaften
 - die gedrückte Taste bei *KeyEvents*
 - die Mausposition bei *MouseEvents*
 - ein *ActionCommand* bei Auswahl eines *JButton*

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

97

ActionEvent

- × Wird erzeugt
 - bei Mouseklick auf einen *JButton*
 - wenn die Return-Taste in einem *TextField* gedrückt wurde
 - bei Auswahl eines Menüpunktes (*JMenuItem*)
 - bei Doppelklick auf einen Listen-Gegenstand
- × Vererbung
 - Object ← EventObject ← AWTEvent ← ActionEvent

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

98

ActionEvent

× Methoden

- Object **getSource()** Ergebnis ist eine Referenz auf das Objekt, von dem das Ereignis erzeugt wurde
- String **getActionCommand()** Liefert einen Befehlstext, der mit dem Ereignis verbunden ist

Der Befehlstext wird in dem Objekt (z.B. JButton), das das Ereignis erzeugt festgelegt.

*Voreingestellt ist die Beschriftung des Objektes. Man kann jedoch unabhängig davon dem Objekt andere "ActionCommands" zuordnen.
→ void **setActionCommand(String ActionCommand)***

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

99

WindowEvent

- × Wird erzeugt bei Mouseklick auf eine Taste im Fensterbalken
- Je nach Aktion werden im Event unterschiedliche Ursachen angegeben
- Drücken des **Exit-Knopfes** wird durch die Ursache **WindowClosing** ausgedrückt
- Weitere Ursachen sind:
 - **windowOpened**
 - **windowIconified**
 - **windowDeiconified**
 - **windowClosed**
 - **windowActivated**
 - **windowDeactivated**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

100

WindowEvent

- ✖ Setzt man in der JFrame-Methode `setDefaultCloseOperation(todo)` den Parameter `todo` auf
 - `Do_Nothing_On_Close`, `Dispose_On_Close` oder `Hide_on_Close`
kann man das Window-Ereignis des Exit-Knopfes abfangen und eigene Anweisungen ausführen lassen.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

101

KeyEvent

- ✖ Ein KeyEvent wird erzeugt
 - wenn man eine Taste der Tastatur drückt
 - Im KeyEvent wird die gedrückte Taste gespeichert. Über eine `get`-Methode kann man diese Information abholen
 - Der KeyEvent geht immer an die Komponente die gerade den Fokus besitzt.
 - Um sicherzustellen, dass die Komponente den Event erhält, bei der man den Listener auch registriert hat, muss sich die Komponente mit der Methode `requestFokus()` den Fokus holen.

Anmerkung:

Wählt man mit der Maus im Navigator eine Bewegungstaste aus, so besitzt danach diese Taste den Fokus.
Soll der KeyEvent ans Hauptfenster sendet werden, muss sich das Fenster nach Auswertung des ActionEvent den Fokus wieder holen.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

102

MouseEvent

- ✖ Ein MouseEvent wird erzeugt
 - wenn man die Maus bewegt
 - wenn man eine Maustaste betätigt
 - Speichert die Position der Mouse innerhalb der Komponente. Mit getX() und getY() kann man diese Position erfragen

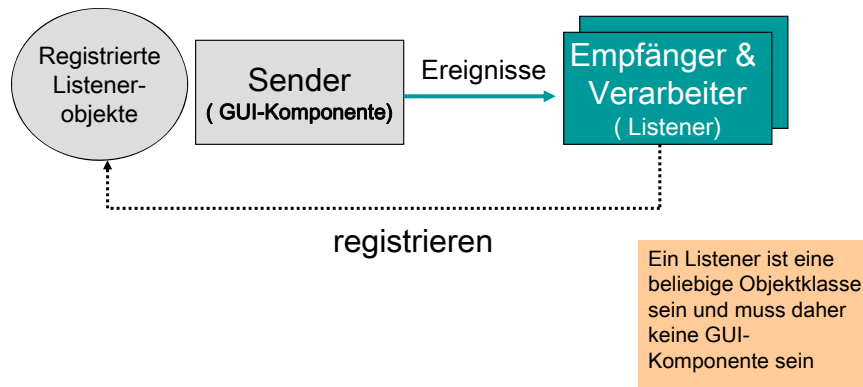
ComponentEvent

- ✖ Lowlevel Ereignis
- ✖ Auslöser ist jede swing-Komponente
- ✖ Wird ausgelöst, falls
 - die Komponente verschoben wurde
 - die Größe der Komponente verändert wurde
 - die Sichtbarkeit der Komponente sich ändert

Hilfestellung:

Auf diesen Event sollte man reagieren, damit man das Navigationsfenster neu positionieren kann, falls das Hauptfenster von DionaRap verschoben oder in der Größe verändert wurde.

Ereignis-Modell in Java



10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

105

Ereignis-Modell in Java

- × **Damit ein Listener-Objekt Ereignisse eines Sender-Objektes behandeln kann, muss es sich beim Sender **registrieren** lassen.**
 - Es ist möglich ein **Listener-Objekt** bei **mehreren unterschiedlichen Sender-Objekten** zu registrieren
 - Es können **mehrere unterschiedliche Listener-Objekte** bei einem **Sender-Objekt** registriert werden.
- × **Das Sender-Objekt sendet Ereignisse an **alle** seine registrierten Listener-Objekte**
 - In welcher Reihenfolge ein Ereignisse zu den unterschiedlichen Listenerobjekten gesendet wird, ist nicht definiert

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

106

Möglichkeiten Listener Objekte zu erzeugen

- Die GUI-Komponente ist selbst auch das Listener-Objekt



→ Beispiel: BeispielActionListenerSenderIstListener.java

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

107

Möglichkeiten Listener Objekte zu erzeugen

- Das Listener-Objekt ist ein eigenständiges Objekt



Wird implementiert als

- public Klasse → Beispiel: BeispielActionListener.java
- interne Klasse
- anonyme Klasse → Beispiel: BeispielActionListenerAnonymeKlasse.java

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

108

Anonyme Klassen

- × Anonyme Klassen sind spezielle lokale Klassen
- × Sie haben keinen Namen
- × Sie erzeugen immer automatisch ein Objekt
 - Klassendeklaration und Objekterzeugung sind zu einem Sprachkonstrukt verbunden
 - Sie werden stets in einer new-Anweisung definiert, wobei hinter dem Klassenbezeichner und der Argumentliste der Klassenrumpf der anonymen Klasse angegeben wird

```
new Thread( new Runnable() { @Override public void run()  
    {  
        for ( int i = 0; i < 10; i++ )  
            System.out.printf( "%d ", i );  
    }  
} ).start();
```



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

109

Anonyme Klassen

- × Wozu braucht man anonyme Klassen?
 - Man kann Sie verwenden, wenn man ein Objekt nur einmal braucht und dieses Objekt nur wenige Aufgaben erledigt.
 - Einsatz in der GUI-Programmierung als Reaktion auf Benutzereingaben (Listener)
 - Multithreading

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

110

Anonyme Klassen

- × Es ist auch möglich eine **anonyme Implementierungen von Interfaces** zu erzeugen.
 - Hier wird bei der Instanziierung der Name des Interface (anstatt des Klassennamen) angegeben
 - Die Angabe des Schlüsselworts **implements** entfällt

```
new ActionListener(){ ... }
```

Name eines Interface

- × Die Parameterliste hinter dem Namen des Interface immer leer sein, da Interfaces keine Konstruktoren besitzen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

111

Implementierung einer Listener-Klasse

- × Zur Implementierung von Listener-Objekten stellt JAVA für Ereignisarten unterschiedliche **Interfaces** zu Verfügung.
- × Beispiel
 - **ActionListener** für ActionEvents
 - Sendertyp : JButton, JTextField
 - **WindowListener** für WindowEvents
 - Sender: JFrame
 - **MouseListener** für MouseEvents
 - Sender: jede Komponente
 - **KeyListener** für KeyEvents
 - Sender: jede Komponente

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

112

Implementierung einer Listener-Klasse

- ✖ Die Objektklasse, die das Listenerobjekt deklariert, implementiert ein Interface und muss daher **alle Methoden** im Interface implementieren.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

113

Implementierung Listener-Klasse am Beispiel MouseListener

```
public class Listing2903 extends JFrame implements MouseListener
{
    ...

    public void mousePressed(MouseEvent event) {}
    public void mouseClicked(MouseEvent arg0) {}
    public void mouseReleased(MouseEvent arg0) {}
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}

    ...
};
```

Bei der Implementierung der Listener Klassen müssen grundsätzlich alle Methoden ausprogrammiert werden, auch, wenn sie nicht benötigt werden

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

114

Registrieren von Listenerobjekten

- × Zum registrieren von Listenerobjekten stellt die Objektklasse des Sender-Objektes Methoden zu Verfügung
 - Beispiel
 - Die Objektklasse JButton stellt zu Verfügung
`void addActionListener(ActionListener)`
 - Die Objektklasse JFrame stellt die Methode zu Verfügung
`void addWindowListener(WindowListener)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

115

Beispiel für Listener-Interfaces

Interface

Deklarierte Methoden

ActionListener

`actionPerformed(ActionEvent e)`

WindowListener

`windowClosing(WindowEvent e)`
`windowOpened(WindowEvent e)`
`windowIconified(WindowEvent e)`
`windowDeiconified(WindowEvent e)`
`windowClosed(WindowEvent e)`
`windowActivated(WindowEvent e)`
`windowDeactivated(WindowEvent e)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

116

Beispiel für Listener-Interfaces

Interface

ComponentListener

KeyListener

Deklarierte Methoden

componentResized(ComponentEvent e)
componentMoved(ComponentEvent e)
componentShown(ComponentEvent e)
componentHidden(ComponentEvent e)

keyTyped(KeyEvent k)
keyPressed(KeyEvent k)
keyReleased(KeyEvent k)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

117

Beispiel – ActionListener

```
...
public class MyActionListener extends JFrame implements ActionListener
{
    // Beim Klicken auf den Button erzeugt die Anwendung einen Laut

    public MyActionListener()
    {
        this. setDefaultCloseOperation(EXIT_ON_CLOSE);
        JButton button= new JButton("Beeper");
        button.addActionListener(this);
        this.getContentPane().add(button);
    }

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }

    public static void main(String[] args){
...

```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

118

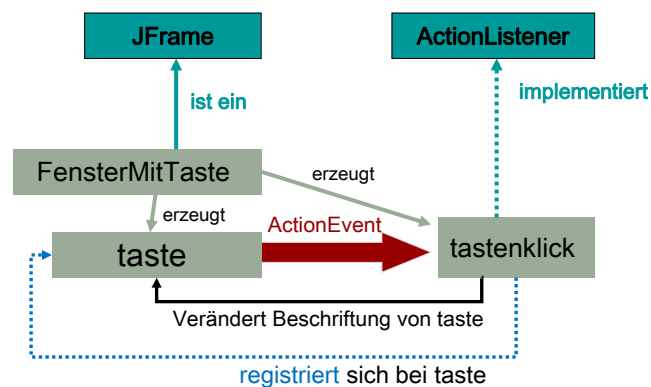
Beispiel – `ActionEvent` – Methode `getActionCommand()`

```
...
public class DoAction extends JFrame implements ActionListener
{
    // Die Anwendung reagiert auf ActionEvents. Diese werden erzeugt durch das Klicken
    // auf den Button. Das ActionCommand zum Button wird in der Konsole ausgegeben

    public DoAction()
    {
        JButton button= new JButton("Button");
        button.setActionCommand("Hallole");
        button.addActionListener(this);
        this.getContentPane().add(button);
        this.pack();
        this.setVisible(true);
    }
    public void actionPerformed(ActionEvent event)
    {
        Object obj = event.getSource();
        if (obj instanceof JButton){
            System.out.println("Action done of: "+event.getActionCommand());
        }
    }
    public static void main(String[] args){ new DoAction(); }
}
```

10 March 2016 Grafische Bedienoberflächen Prof. S. Keller 119

Beispielprogramm für `ActionEvent`



→ Beispielprogramm FensterMitTaste

Beispiel WindowListener

Ein Fenster, reagiert auf den Exit-Knopf

× Java-Objektklasse

- **abgeleitet** von **JFrame**
 - Groesse, Titel und Hintergrundfarbe vorgeben
 - DefaultCloseOperation wird auf Tue-Nichts gestellt
- enthält die Methode **main()**
- **implementiert WindowListener-Interface**
 - Implementierung aller Methoden von WindowListener
 - reagiert nur auf Ereignis **windowClosing** (alle anderen Methoden haben einen leeren Rumpf)
- **registriert** sich selbst als WindowListener-Objekt

JAVA-Klasse

Frame
&
Listener



Window
Event

→ Beispielprogramm ErstesFensterIstListener

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

121

Beispiel

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

public class dezimal2dual extends JFrame implements WindowListener
{
    public dezimal2dual()
    {
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        this.setSize(400,400);
        this.setBackground(Color.blue);
        this.setTitel("Dezimal nach Dual Umrechnung");
        this.setVisible(true);
        this.addWindowListener(this);}

    public static void main( String arg[] )
    {
        new dezimal2dual();
    }

    // Interface implementieren

    public void windowClosing(WindowEvent e)
    {
        System.exit(0);}

    public void windowActivated(WindowEvent e) {}

    ..... // alle restlichen Methoden des Interfaces implementieren
}

```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

122

Adapter-Klassen

- ✖ Zur Vereinfachung stellt Java abstrakte **Adapterklassen** für jedes Listener-Interface zu Verfügung
- ✖ Eine Adapterklasse enthält für **alle Methoden** des Interfaces **leere Rümpfe** (also: tue nichts)
- ✖ Die selbst definierte Listenerklasse in der Anwendung braucht unter Verwendung der Adapterklassen nur noch die Methoden zu implementieren, auf die auch reagiert werden soll.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

123

Beispiel: Adapter Klasse für **MouseListener**

```
... public class Listing2903 extends MouseAdapter
{
    //nur gewünschte Methoden des MouseAdapters müssen eingefügt
    werden
} ...
```

Sie ersparen sich das
implementieren aller Methoden
eines Listeners.

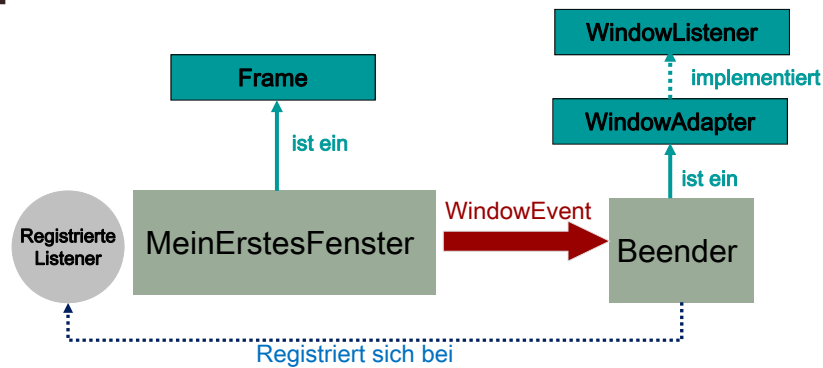
Nur noch die gebrauchten
Methoden müssen implementiert
werden.

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

124

BeispielProgramm Fenster reagiert auf den Exit-Knopf



→ Beispielprogramm ErstesFensterMitListener

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

125

Kommunikation der Listenerobjekte mit der Anwendung

- ✖ Wie kann ein Listenerobjekt auf die Komponenten der Oberfläche zugreifen, wenn es in einer eigenen Klasse realisiert ist?
 - parent-Hierarchie bis JFrame abarbeiten
 - Dann über get/set-Methoden auf die GUI-Komponenten zugreifen
 - Objekte dem Listener per Parameterübergabe bekannt machen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

126

Hierarchischer Aufbau einer GUI

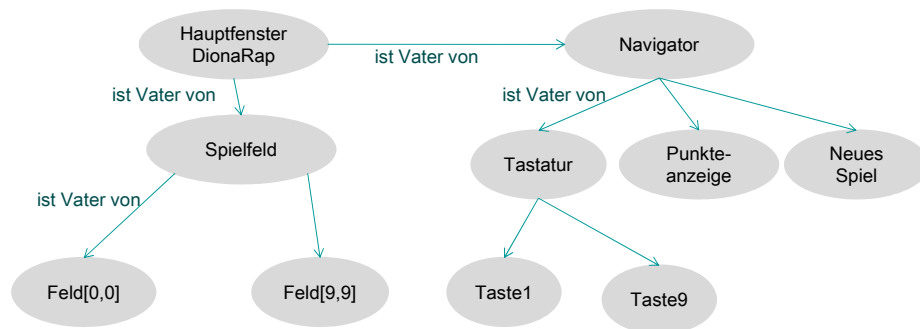
- ✖ Durch Ineinanderschachteln von Containern und platzieren von Komponenten in einem Container wird eine **Baumstruktur** erzeugt
- ✖ Java verwaltet diese Baumstruktur

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

127

Baumstruktur von DionaRap (Logisch)

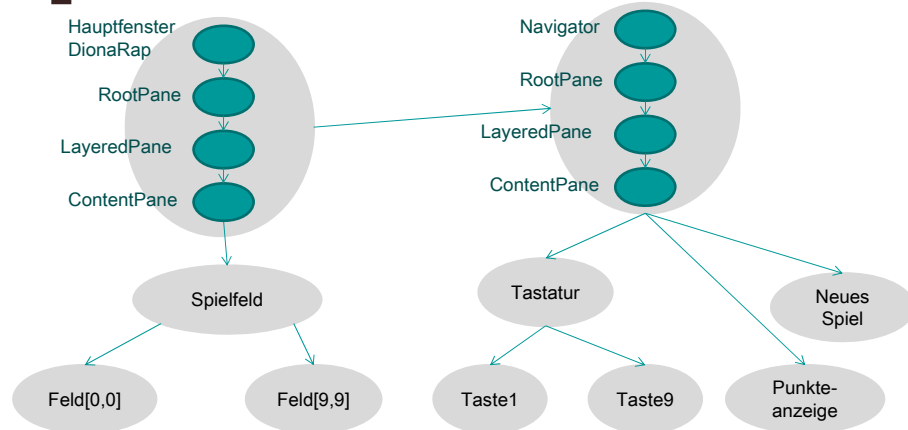


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

128

Baumstruktur von DionaRap (in swing)



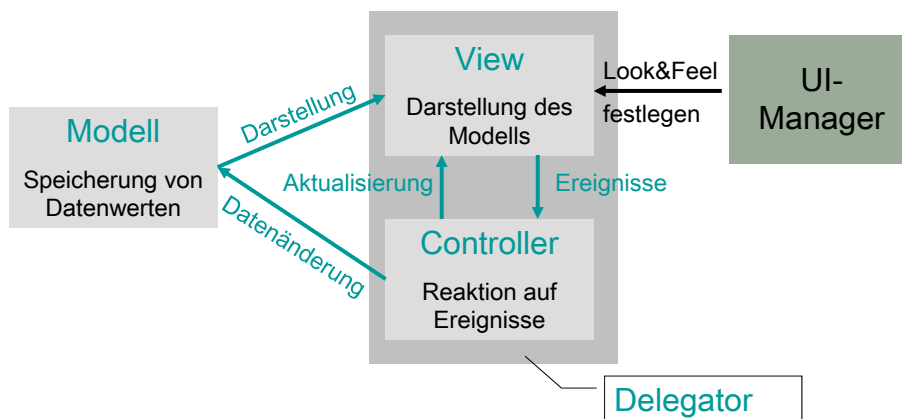
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

129

Swing

MVC-Programmiermodell



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

130

User definierte Events

✖ Wie kann man sich eigene Eventtypen definieren ?

■ Beispiel:

- Spielelogik von DionaRap definiert DionaRapChangedEvent und einen DionaRapListener

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

131

User definierte Events

✖ 1. Schritt

- ➔ Für den neuen Eventtyp eine eigen Klasse definieren.

■ Diese Klasse ableiten von EventObject

```
import java.util.EventObject;  
public class MyEvent extends EventObject
```

- In der Klasse dann Eigenschaften für den Event und Methoden festlegen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

132

User definierte Events

2. Schritt

Ein Listenerinterface definieren

```
public interface MyListener extends EventListener
{
    public void MyListenerMethod(MyEvent e);
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

133

User definierte Events

× 3. Schritt

- Bei dem Objekt, das den Event auslösen soll eine dynamische Liste anlegen, in der sich Listenerobjekte registrieren können

```
private Vector<MyListener> subscribers = new
    Vector();
```

- Eine Methode zum registrieren von Listenerobjekten implementieren

```
public void addMyListener(MyListener listener)
{ ..... }
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

134

User definierte Events

→ 4. Schritt

- Soll Aufgrund eines Ereignisses der Event ausgelöst werden, ein Eventobjekt instanziiieren und dieses Objekt an alle registrierten Listener senden

```
for( MyListener listener : this.subscribers )  
{  
    listener.MyListenerMethode(new MyEvent());  
}
```

Bilder zum dekorieren der GUI verwenden

- ✖ Bilder, die klein sind und auf der GUI nur zur Dekoration von GUI-Komponenten benutzt werden sind **Icons**
 - Icon können statt Text bei Labels oder Buttons verwendet werden
- ✖ Größere Bilder wie Fotos, die im Programm sowohl dargestellt als auch verändert werden können, sind **Images**
 - Ein Image muss auf eine Zeichenfläche gezeichnet werden (*dieses Thema wird später behandelt*)

Anzeigen von kleinen Bildern - Icons

- × Eine sehr einfache Möglichkeit kleine Bilder in Java darzustellen ist die Verwendung der Objektklasse `JLabel` oder `JButton` in Verbindung mit einem `Icon`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

137

Interface Icon

- × Ein `Icon` wird in swing als `Interface` definiert

- Zur Erzeugung eines `Icon` muss eine Objektklasse programmiert werden, die folgende Methoden des Interface implementiert:

```
int getIconHeight()  
int getIconWidth()  
void paintIcon(Component c, Graphics g, int x, int y)
```

Um ein `Icon` darstellen zu können wird eine GUI-Komponente wie z.B. ein `Label` oder ein `Button` benötigt, in die an der Position `x, y` ein Bild gezeichnet wird.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

138

Klasse `ImageIcon`

- ✖ Java stellt eine Objektklasse `ImageIcon` zu Verfügung, die das `Icon` Interface implementiert.
 - Das Bild kann über einen Dateinamen bzw. URL angegeben werden
 - Beim `ImageIcon` werden die `Bilddaten` sofort von der Platte in den Arbeitsspeicher geladen.

Achtung !

Das Objekt `ImageIcon` ist erst dann vorhanden, wenn das Bild vollständig geladen ist.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

139

Ein `ImageIcon` erzeugen

- Ein Icon erzeugen
 - Bilddaten werden aus einer lokalen Datei geladen
`new ImageIcon(String dateiname)`
 - Bilddaten werden über das Netzwerk geladen
`new ImageIcon(URL adresse)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

140

Ein Label mit einem Icon dekorieren

× Label mit einem Icon erzeugen

- `new JLabel(Icon image)`
- `new JLabel(Icon image, int horizontalAlignment)`

× Label mit Text und Bild erzeugen

- `new JLabel(String text, Icon icon, int horizontalAlignment)`
- Für die Ausrichtung sind folgende Konstanten in JLabel möglich:
`LEFT, CENTER, RIGHT`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

141

Anzeigen von kleinen Bildern

- Wird Bild und Text im Label verwendet, kann über die folgenden Methoden die Position von Text und Bild beeinflusst werden.

■ Horizontale Position des Textes relativ zum Bild einstellen

void `setHorizontalTextPosition`(int TextPosition)

Konstanten für TextPosition in JLabel: `TOP, BOTTOM, CENTER`

■ Abstand zwischen Bild und Text einstellen

void `setIconTextGap`(int Abstand)

10 March 2016

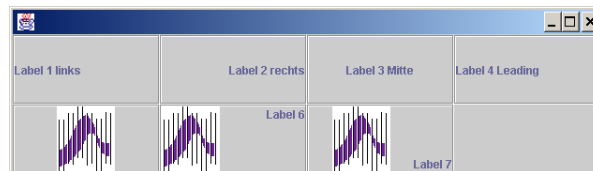
Grafische Bedienoberflächen
Prof. S. Keller

142

Anzeigen von kleinen Bildern

Beispiel

```
JLabel label6;  
label6=new JLabel("Label 6",new  
    ImageIcon(fh.gif"),JLabel.LEFT);  
label6.setVerticalTextPosition(JLabel.TOP);  
label6.setIconTextGap(50);  
zeichenflaeche.add(label6);
```



→ siehe Moodle *labelbeispiel.java*

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

143

Taste mit Bild ausstatten

× Konstruktoren

JButton(Icon Ikone) Button mit einem kleinen Bild dekorieren

JButton (String Text, Icon Ikone) Taste mit Text und Bild darstellen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

144

Beispiel Taste mit Bild ausstatten

```
public class TasteMitBild extends JFrame{

    Icon bild;
    JLabel label;
    JButton button;

    public TasteMitBild(){
        label = new JLabel();           /* Label ohne Beschriftung erzeugen
                                         // ein Icon ( kleines Bild ) erzeugen
        bild= new ImageIcon(System.getProperty("user.dir")+"\\\\"+"bild1.gif");
        label.setIcon(bild);           /* Label mit Bild versehen */
        this.getContentPane().add(label, BorderLayout.CENTER;

        button= new JButton(bild);      /* Taste mit einem Bild erzeugen */
        this.getContentPane().add(button, BorderLayout.EAST);
    }
    public static void main(String[] args){...};
}
```

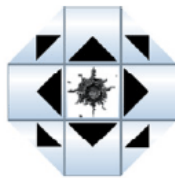
10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

145

Erweiterte Eigenschaften von Fenstern

- ✖ Ab JDK 1.6 update 10 können Fenster in swing transparent und in beliebiger Form dargestellt werden.



Navigationsfenster
von DionaRap

- ✖ Dazu wird ab der Java Version 7 in der Fensterklasse folgende Methoden zu Verfügung gestellt:
 - <fenster>.setShape(Shape form)
 - <fenster>.setUndecorated(true)
 - <fenster>.setOpacity(float alphawert)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

146

Beispiel: Form eines Fensters festlegen

- × **Shape ist ein Interface. Die Klasse Polygon implementiert das Interface Shape**

- Das Interface Shape deklariert eine Methode `getBounds()`. Damit erhält man das umschliessende Rechteck der Form.
- Das umschliessende Rechteck wird für die Einstellung der Fenstergröße benötigt

```
public class Dreieck extends Polygon
{
    Dreieck()
    { super();
      // Eckpunkte des Polygons festlegen
      this.addPoint(0, 0);
      this.addPoint(200, 0);
      this.addPoint(100,100);
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

147

Beispiel: Form eines Fensters festlegen

```
public class WindowMitShape extends JFrame
{
    WindowMitShape()
    { this.setUndecorated(true); // Fensterbalken inaktivieren

      Dreieck dreieck = new Dreieck(); // Shape für das Fenster erzeugen
      this.setShape(dreieck); // Shape dem Fenster zuweisen

      // Das ist die Größe des Fensters und wird später für setSize() gebraucht
      Rectangle umschliessendesrechtecke= dreieck.getBounds();

      // Fenster mittig auf Bildschirm platzieren
      this.setLocationRelativeTo(null);

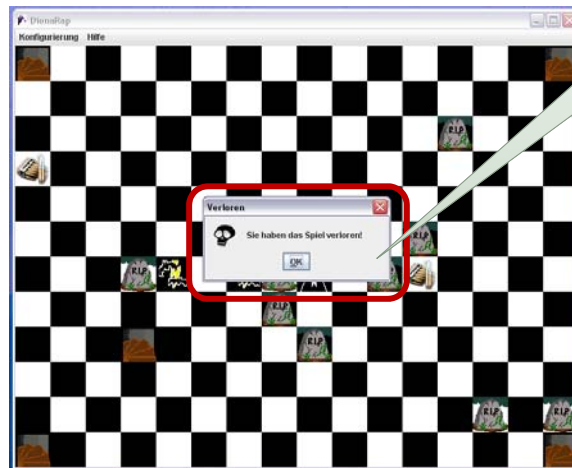
      this.setVisible(true);
      // Fenster muss so groß sein, damit das Dreieck vollständig dargestellt werden kann
      this.setSize(umschliessendesrechtecke.width, umschliessendesrechtecke.height);
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

148

DionaRap mit Dialogfenstern erweitern



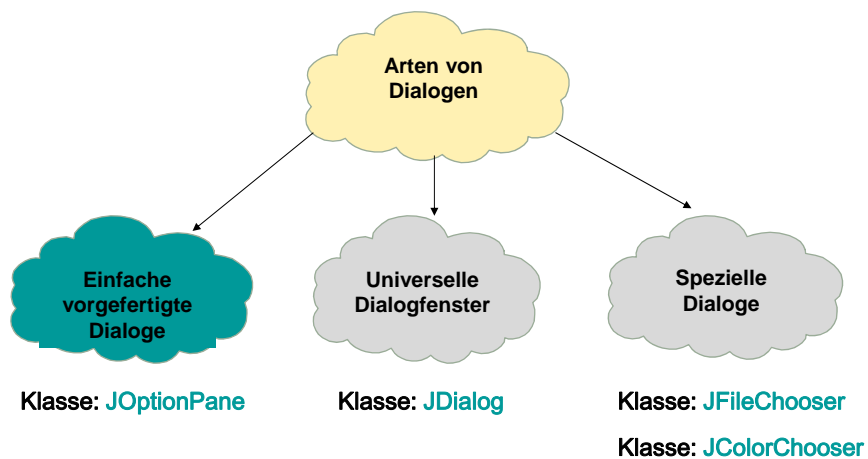
Dialogfenster

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

149

Dialoge in swing



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

150

- ✖ Über die Klasse **JOptionPane** können einfache, vordefinierte Dialoge erzeugt und verwendet werden
- ✖ Ein **JOptionPane** ist ein Popup-Fenster mit einem Titel und 4 Bereichen
 - ➔ Bereich für Anzeige einer Ikone
Vordefinierte Ikonen:
 - **INFORMATIONAL, QUESTION, WARNING, ERROR**
 - ➔ Nachrichtebereich
 - Textnachricht oder Java-Objekt mit Anzeige
 - ➔ Eingabebereich
 - Wahlweise Textfeld, Listbox, Combobox
 - ➔ Tasten-Bereich
 - Auswahl einer Taste beendet den Dialog.
 - Vordefinierte Tasten: **OK, Cancel, YES, NO**

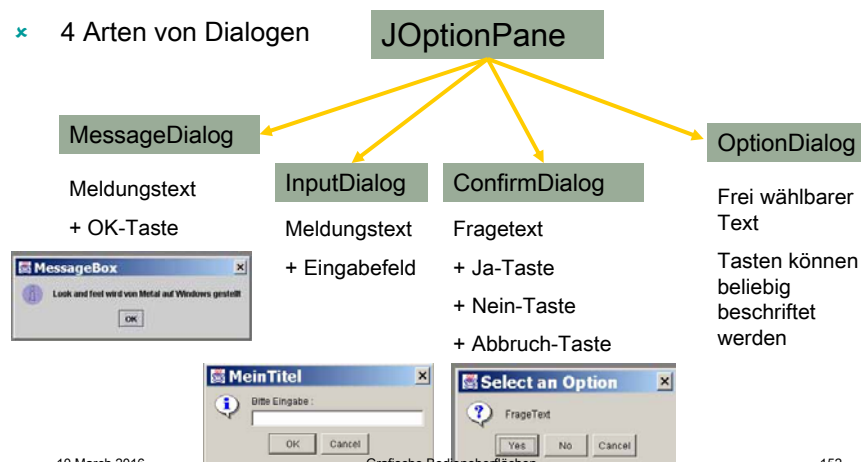


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

151

- ✖ 4 Arten von Dialogen



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

152

Klasse JOptionPane

- × Erzeugen eines Dialogs
 - Ein vorgefertigter Dialog wird immer in einem Popup-Fenster dargestellt.
 - Die Klasse stellt Klassen-Methoden zu Verfügung, mit der man einen Dialog erzeugen und anzeigen kann
 - Das Popup-Fenster des Dialog kann entweder als modaler Dialog oder als internes Fenster dargestellt werden
 - `JOptionPane.showXXXDialog()`
 - `JOptionPane.showInternalXXXDialog()`
 - XXX: Confirm / Message / Input / Option

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

153

Klasse JOptionPane

Beispiele:

`showConfirmDialog(Component parent, Object message)`

`showMessageDialog(Component parent, Object message, String title, int messageType)`

`showMessageDialog(Component parent, Object message, String title, int messageType, Icon eigenesBild)`

Für den Parameter `messageType` stehen in der Klasse `JOptionPane` Konstanten zu Verfügung

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

154

Beispiel

JOptionPane

```
import javax.swing.*;

public class JOptionPaneDialoge extends JFrame
{
    public JOptionPaneDialoge(){

        JOptionPane.showMessageDialog( this, "Willkommen in der Java - Welt" );
        JOptionPane.showMessageDialog( this, "Alles klar" );
        // Dialog mit eigenem Bild
        JOptionPane.showMessageDialog(this,"Die Vorlesung hat gefallen", "Antwort
auf Frage",JOptionPane.PLAIN_MESSAGE, new ImageIcon("fh.gif"));
        // Dialog zum Auswählen aus einer vordefinierten Liste
        String[] liste = {"Informatik", "Maschinenbau", "keine Ahnung", "ändert
sich ständig" };
        String studgng = (String) JOptionPane.showInputDialog(
this,"Studiengänge","Bitte den Studiengang wählen",
JOptionPane.QUESTION_MESSAGE,null,liste,liste[1]);
    }

    public static void main( String[] args ){ new JOptionPaneDialoge();};
}
```

Die Anwendung erzeugt eine Reihe von vorgefertigten, einfachen Dialogen.

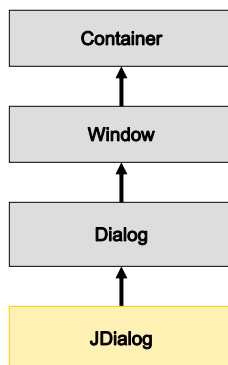
10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

155

Universelle Dialoge

Swing Top-Level-Container JDialog



- ✖ JDialog ist abgeleitet von der awt-Klasse Dialog und damit eine heavyweight-Komponente
- ✖ Gehört in swing zu den Top-Level-Container und setzt sich analog einem JFrame aus mehreren Containern zusammen
- ✖ Alle Oberflächen-Komponenten müssen daher in den ContentPane des Dialoges eingefügt werden
- ✖ Die Reaktion auf den exit-Knopf kann entweder über die Methode `JDialog.setDefaultCloseOperation()` oder über einen registrierten `WindowListener` festgelegt werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

156

Swing- Top-Level-Container JDialog

- ✖ Eigenschaften
 - modal / nicht modal
 - modal: Anwendung ist gesperrt, bis der Dialog beendet wurde
 - nicht modal: Der Dialog läuft in einem eigenen Thread. Die Anwendung wird nicht blockiert.
- ✖ Bei WindowClosing (Exit-Knopf) soll das Fenster zerstört werden, die Anwendung muss aber weiter laufen.
 - Methode `<JDialog>.dispose()` im Window-Listener
 - Konstante `DISPOSE_ON_CLOSE` in der JDialog-Methode `setDefaultCloseOperation()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

157

Swing- Top-Level-Container JDialog

- ✖ Konstruktoren
 - `JDialog()`
 - `JDialog(JFrame owner), JDialog(JDialog owner)`
 - `JDialog(JFrame, boolean modal), JDialog(JDialog owner, boolean modal)`
 - `JDialog(JFrame, String titel), JDialog(JDialog owner, String titel)`
 - `JDialog(JFrame, String titel, boolean modal), JDialog(JDialog owner, String titel, boolean modal)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

158

Beispiel : Einen `JDialog` erzeugen und darstellen

```
...
public class MyDialog extends JFrame
{
    JDialog dialog;
    JLabel label;

    public MyDialog()
    {
        dialog = new JDialog(this, "Hello", false);
        label= new JLabel("Hello World", Label.CENTER);
        dialog.add(label);           // Text in Dialog einfügen
        dialog.setSize(100,100);    // Größe Dialogfenster setzen
        dialog.setLocation(100,100); // Position Dialogfenster
        dialog.setVisible(true);     // Dialogfenster sichtbar machen
        dialog.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

    public static void main(String[] args){ ... }
}
```

Es wird ein modales
Dialogfenster erzeugt,
ausgestattet mit einem Label.

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

159

Beispiel : Eine eigene `Dialogklasse` erstellen

- × Klasse `BildAnzeige` wird abgeleitet von `JDialog` und
 - enthält ein Label
 - Dieses zeigt ein kleines Bild an
- × Der Dialog muss einem Vaterfenster zugeordnet werden. Dies kann über den Aufruf eines Konstruktors von `JDialog` erfolgen. Daher muss als erste Methode im Konstruktor von `Meldung` die Methode `super(...)` verwendet werden.
 - Mit `super` kann dann der gewünschte Konstruktor von Dialog verwendet werden
 - z.B. `super(JFrame Owner, String Titel)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

160

Beispiel : Eine eigene Dialogklasse erstellen

```
public class MyDialog extends JDialog
{
    ImageIcon bild;
    JFrame parent;

    public MyDialog(JFrame parent)
    {
        super(parent, "Hello", false);
        bild = new ImageIcon("fh.gif");
        JLabel label = new JLabel(bild);
        this.add(label); // Label in Dialog einfügen
        this.setSize(bild.getHeight(), bild.getWidth());
        this.setLocation(100, 100); // Position Dialogfenster
        this.setVisible(true); // Dialogfenster sichtbar machen
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

161

JOptionPane positionieren

- ✖ JOptionPane ist selbst kein Fenster
- ✖ `JOptionPane.showXXXDialog()` erzeugt ein JOptionPane und stellt es in einem modalen JDialog dar
- ✖ Das Dialogfenster wird immer mittig über dem Vaterfenster positioniert
 - ➔ Eine Angabe der Position ist nicht möglich

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

162

JOptionPane positionieren

- ✖ Um ein JOptionPane positionieren zu können, muss man die Komponente instanziiieren und nachträglich in einem JDialog platzieren
- ✖ Diesen JDialog kann man dann positionieren

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

163

JOptionPane positionieren

```
JOptionPane messagebox = new JOptionPane();  
// Typ ist MessageDialog  
messagebox.setMessageType(JOptionPane.INFORMATION_MESSAGE);  
// Text der Meldung  
messagebox.setMessage(Message);  
// JDialog für JOptionPane erzeugen  
JDialog messagedialog = messagebox.createDialog(this, "Ergebnis");  
  
// Pop-upfenster positionieren  
messagedialog.setLocation(this.getLocation().x+this.getWidth()/2-  
    messagedialog.getWidth()/2, this.getLocation().y+this.getHeight()+10);  
  
messagedialog.setVisible(true);
```

10 March 2016

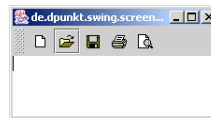
Grafische Bedienoberflächen
Prof. S. Keller

164

DionaRap um eine Toolbar erweitern

- ✖ JToolBar ist ein Container mit **BoxLayout**
- ✖ Eine Toolbar wird verwendet, um häufig gebrauchte Funktionen zu gruppieren
 - ➔ Die Toolbar enthält daher in der Regel mit Bildern dekorierte Buttons. Über diese Buttons können dann die Funktionen aufgerufen werden

Es können jedoch auch beliebige andere GUI-Komponenten in eine Toolbar gelegt werden z.B. ein Textfeld für eine Such-Funktion



Quelle:
http://www.dpunkt.de/java/Programmieren_mit_Java/Oberflaeche/aechenprogrammierung/23.html

10 March 2016

Grafische Bedienoberflächen
 Prof. S. Keller

165

DionaRap um eine Toolbar erweitern



Toolbar von
 DionaRap

- ✖ Die Toolbar von DionaRap enthält neben einer Aktion „Neues Spiel starten“ vorwiegend Anzeige-Objekte zur Darstellung des Spielzustandes, wie Punktestand, Spielfortschritt und vorhandene Munition.
 - ➔ Punktestand wird in einem **JTextField** angezeigt
 - ➔ Spielfortschritt wird in einer **JProgressbar** angezeigt
 - ➔ Vorhandene Munition – hier wird für die Munition Bilder in ein Panel gezeichnet (→ wird erst in Übungsblatt 5 realisiert)

10 March 2016

Grafische Bedienoberflächen
 Prof. S. Keller

166

Klasse JToolBar

- ✖ EineToolBar kann in die Bereiche NORTH, SOUTH, WEST oder EAST gelegt werden.

- Zur Laufzeit kann der Anwender die Toolbar an eine gewünschte Position frei bewegen. Bewegt er die Toolbar aus dem Fenster heraus, wird die Toolbar in einem eigenen Dialogfenster dargestellt.



- Um eine Verschiebung in die 4 Bereiche realisieren zu können, muss für das Fenster das `BorderLayout` eingestellt sein.
- Um eine Bewegung vom Benutzer zu deaktivieren, stellt `JToolBar` die Methode `setfloatable(boolean)` zu Verfügung.
 - Wird `false` übergeben, so ist die Werkzeugleiste an der vorgegebenen Position fest und kann vom Benutzer nicht mehr verschoben werden.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

167

LayoutManager BorderLayout

- ✖ Bisherigen Layout Manager BorderLayout, FlowLayout und GridLayout sind Bestandteil des awt
 - Werten keine Größenangaben wie Maximale Größe, bevorzugte Größe, Minimale Größe und auch keine Ausrichtung der Komponente auswerten.
- ✖ Swing ermöglicht gegenüber dem AWT zusätzliche Layout Manager.
 - Diese Layoutmanager sind intelligenter und werten alle Eigenschaften einer Komponente zur Platzierung aus
 - BorderLayout, SpringLayout
- ✖ LayoutManager von Drittanbieter
 - MigLayout (<http://www.miglayout.com/>)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

168

BoxLayout

- ✖ Ein hilfreicher Layout Manager von swing ist das **BoxLayout**
 - Die Komponenten können entweder **horizontal** in Reihen oder **vertikal** in Spalten angeordnet werden
 - Konstruktor:

`BoxLayout(Container BoxLayoutContainer, int ausrichtung)`
 - Klassenkonstanten für die Ausrichtung: **X_AXIS, Y_AXIS**

Tipp

→ siehe Oracle Doku: [How to use BoxLayout](#)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

169

Box-Layout

- ✖ **Anordnung von Komponenten mit gleicher Ausrichtung**
 - Bei horizontalem BoxLayout werden die Komponenten entsprechend der vertikalen Ausrichtung angeordnet
 - Bei vertikalem BoxLayout werden die Komponenten entsprechend dem horizontalen Ausrichtung angeordnet



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

170

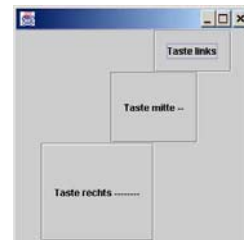
Box-Layout

✖ Anordnung von Komponenten mit unterschiedlicher Ausrichtung

→ Alle Komponenten werden in Bezug auf eine errechnete **Mittellinie** ausgerichtet

■ Bei einem vertikalen BoxLayout werden alle Komponenten

- mit linksbündiger Ausrichtung mit ihrer linken Seite auf die Mittellinie platziert
- Mit rechtsbündiger Ausrichtung mit ihrer rechten Seite auf die Mittellinie platziert
- Mit zentrierter Ausrichtung mit Ihrem Mittelpunkt auf die Mittellinie platziert



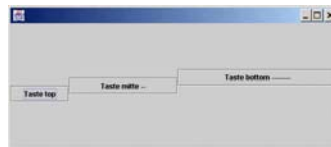
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

171

Box-Layout

■ Beispiel für ein horizontales BoxLayout



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

172

Beispiel – Box Layout

```
... public class BeispielBoxLayout extends JFrame
{
    public BeispielBoxLayout(){

        JPanel panel =new JPanel();
        panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS)); /* Panel BoxLayout zuweisen */

        JButton button1=new JButton("Button1");           /* 2 Tasten erzeugen */
        button1.setPreferredSize(new Dimension(100,100));
        button1.setAlignmentX(JComponent.CENTER_ALIGNMENT);
        button1.setAlignmentY(JComponent.TOP_ALIGNMENT);
        JButton button2=new JButton("Button2");
        button2.setPreferredSize(new Dimension(50,50));
        button2.setAlignmentX(JComponent.CENTER_ALIGNMENT);
        button2.setAlignmentY(JComponent.CENTER_ALIGNMENT);

        panel.add(button1);                                /* Button1 ins Panel legen */
        panel.add(button2);                                /* Button2 darunter anordnen */

        Container cont=this.getContentPane();
        cont.add(panel);                                    /* Panel ins Fenster einfügen */
    }
    ...
}
```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

173

Swing-Klasse Box

- ✖ Die Klasse **Box** ist ein Container mit voreingestelltem Box-Layout
 - ➔ Vereinfacht das Arbeiten mit einem BoxLayout
 - ➔ Konstruktor:
 - `public Box(int plazierungsrichtung)`
 - Beispiel: `new Box(BoxLayout.X_AXIS);`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

174

Swing-Klasse Box

- × Um Abstände und Lücken zwischen den sichtbaren Komponenten herzustellen stellt die Klasse Box über Klassen-Methoden **unsichtbare Komponenten** zu Verfügung

- **glue** (unsichtbare Komponente mit variabler Größe)
Die Größe wird vom LayoutManager entsprechend der Fenstergröße und der preferredSize aller sichtbaren Komponenten berechnet

```
JComponent Box.createHorizontalGlue(),  
Box.createVerticalGlue()
```

- **strut** (unsichtbare Komponente mit fester Größe)

```
JComponent Box.createHorizontalStrut(int breite),  
Box.createVerticalStrut(int höhe)
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

175

Beispiel – Box Layout

```
... public class BeispielBoxLayout extends JFrame  
{  
    public BeispielBoxLayout(){  
        JPanel panel =new JPanel();  
        panel.setLayout(new BorderLayout(panel,BoxLayout.Y_AXIS)); /* Panel BorderLayout zuweisen */  
        JButton button1=new JButton("Button1"); /* 2 Tasten erzeugen */  
        button1.setPreferredSize(new Dimension(100,100));  
        button1.setAlignmentX(JComponent.CENTER_ALIGNMENT);  
        button1.setAlignmentY(JComponent.TOP_ALIGNMENT);  
        JButton button2=new JButton("Button2");  
        button2.setPreferredSize(new Dimension(50,50));  
        button2.setAlignmentX(JComponent.CENTER_ALIGNMENT);  
        button2.setAlignmentY(JComponent.CENTER_ALIGNMENT);  
  
        panel.add(button1); /* Button1 links Panel legen */  
        panel.add(Box.createVerticalGlue()); /* variabler Leerraum erzeugen und  
        darunter legen */  
        panel.add(button2); /* Button2 unter den Leerraum legen */  
  
        Container cont=this.getContentPane(); /* Panel ins Fenster legen  
        cont.add(panel);  
    }  
    ...  
}
```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

176

Beispiel – Klasse Box

```
public class MyBox extends JFrame
{
    Container c;
    Box myBox;

    /* Die Klasse Box hat voreingestellt das BoxLayout und vereinfacht
       das Arbeiten mit diesem LayoutManager. Zusätzlich können
       Abstände zwischen den einzelnen Komponenten erzeugt werden. */

    public MyBox(){

        myBox= Box.createHorizontalBox(); /* Panel mit horizontalem BoxLayout erzeugen */
        myBox.add(Box.createGlue());
        myBox.add(new JButton("Left"));
        myBox.add(Box.createGlue());
        myBox.add(new JButton("Middle"));
        myBox.add(Box.createGlue());
        myBox.add(new JButton("Right"));

        c= this.getContentPane();
        c.add(myBox);
        this.setSize(400,400);
        this.setVisible(true);
    }

    public static void main(String[] args){ new MyBox(); }
}
```

10 March 2016

Grafische Bedienoberflächen Prof. S. Keller

177

Beispiel Toolbar

```
public class MeineToolbar extends JToolBar
{
    MeineToolbar(String titel)
    {
        super(titel);
        JButton action;

        for ( int i=0; i<3; i++)
        {
            action = new JButton("action"+i);
            action.addActionListener( new ActionListener()
            {
                public void actionPerformed(ActionEvent event)
                {
                    System.out.println(event.getActionCommand());
                }
            });
            this.add(action);
        }
    }
}
```

Titel des
Dialogfensters

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

178

Beispiel Toolbar

```
public class BeispielToolbar extends JFrame
{
    BeispielToolbar()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Beispiel Toolbar");

        MeineToolbar toolbar = new MeineToolbar("Das sind Ihre
                                                Tools");

        // Toolbar ganz oben in North des Fensters legen
        this.add(toolbar, BorderLayout.PAGE_START);

        this.setSize(400,200);
        this.setVisible(true);
    }
}
```

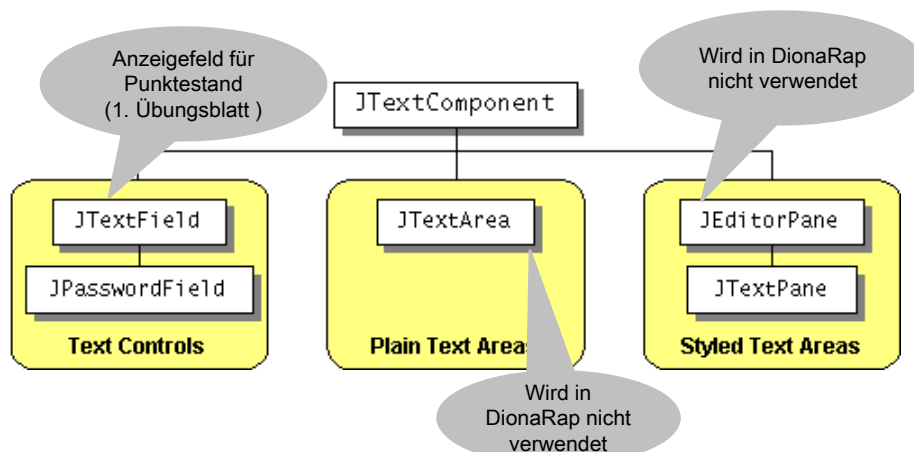
Im Bereich
Nord
anordnen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

179

Typen von Textkomponenten

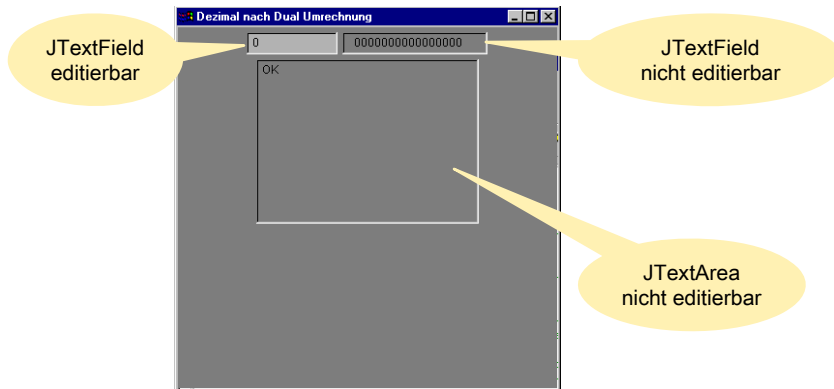


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

180

Einfache Textfelder in swing JTextField und JTextArea



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

181

Klasse JTextField swing

- × Einzeiliges Textfeld
- × voreingestellt ist es editierbar
- × es kann auch als nicht editierbares Ausgabefeld benutzt werden
 - `<textfield>.setEditable(false)`
- × die Breite ist einstellbar und wird angegeben als Anzahl von Spalten
 - `<textfield>.setColumns(int AnzahlSpalten)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

182

Klasse JTextField

× Konstruktoren

- JTextField(),
- JTextField(String Text),
- JTextField(int Breite), JTextField(String Text, int Breite)

× wichtigste Methoden

- | | |
|---|------------------------------|
| → String getText() | Text auslesen |
| → setText(String Text) | Text darstellen |
| → setColumns(int Breite) | Breite in Anzahl von Zeichen |
| → setEditable(boolean) | Voreinstellung ist true |
| → setEchoChar(char echozeichen) | |
| → setToolTipText(string Text) | Tooltip verwenden |
| → setFont(Font f) | Font für Text einstellen |
| → setForeground(Color Textfarbe) | Textfarbe setzen |
| → setBackground(Color Hintergrundfarbe) | Hintergrundfarbe setzen |

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

183

Klasse JTextArea

- × Mehrzeiliges Textfeld
- × Voreingestellt editierbar
- × kann auch als nicht editierbarer Ausgabebereich benutzt werden
- × Anzahl von Zeilen und Spalten ist einstellbar
- × Text in dem Feld wird als **Document** gespeichert
- × Sollte das Document nicht vollständig dargestellt werden können, kann das Textfeld in ein **JScrollPane** eingefügt werden.
 - Durch das JScrollPane erscheinen Rollbalken, falls Sie benötigt werden.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

184

Klasse JTextArea

swing

- abgeleitet von der Klasse `JTextComponent`
- implementiert das Interface `Document`
 - `Document` übernimmt Verwaltung, Bearbeitung und Speicherung der Textdaten
- Bei jeder Texteingabe oder Textänderung erzeugt das Document einen `DocumentEvent`

*Anmerkung:
Events werden später im Kapitel „Ereignisse in Java“ behandelt*

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

185

TextArea

* Konstruktoren

- `TextArea()`, `TextArea(String text)`, `TextArea(int Zeilen, int Spalten)`

* wichtigste Methoden

- | | |
|---|--|
| → <code>append(String text)</code> | Text ans Ende anfügen |
| → <code>insert(String text, int position)</code> | Text an eine best. Position einfügen |
| → <code>setColumns(int spalten), setRows(int zeilen)</code> | Breite und Anzahl Zeilen festlegen |
| → <code>setEditable(boolean)</code> | Voreinstellung ist true |
| → <code>setText(String Text)</code> | Textinhalt festlegen |
| → <code>getText()</code> | Text auslesen |
| → <code>setToolTipText(string Text)</code> | |
| → <code>setLineWrap(boolean)</code> | Voreinstellung ist false |
| → <code>setBackground(Color)</code> | |
| → <code>getDocument()</code> | Liefert das Dokument, das den Text verwaltet.
Wird benötigt, um auf ein <code>DocumentEvent</code> zu reagieren |

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

186

Interface Document

- × Package javax.swing.text
- × Ein Document ist ein Container für Text.
 - Dieser Text kann ein einfacher Text sein, wie z.B. im JTextArea
 - oder aber strukturierter und formatierter Text wie z.B. HTML
- × Methoden für die Verwendung einfacher Texte
 - `int getLength()` Liefert die Gesamtzahl der Zeichen im Dokument
 - `String getText(int offset, int länge)` Liefert den einen Text der angegebenen Länge, beginnend ab der Position offset
 - `void remove(int offset, int länge)` Löscht einen Text der angegebenen Länge, beginnend ab der Position offset im Dokument

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

187

Klasse JScrollPane

swing

- × Ein Panel, das mit einer swing swing-Komponente verbunden wird, die als Anzeige für Daten verwendet wird wie z.B. Texte
 - Nur lightweight-Komponenten können mit der JScrollPane verbunden werden
- × Können die Daten nicht vollständig in dem Panel angezeigt werden, wird nur ein Ausschnitt (`view`) angezeigt. Über vertikale und/oder horizontale Rollbalken kann der Ausschnitt bewegt werden (`Scrollen`)
- × Ein Tutorial zur Verwendung von JScrollPane findet man unter <http://java.sun.com/docs/books/tutorial/uiswing/components/scrollpane.html>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

188

Klasse JScrollPane Beispiel

```
public class BeispielScrollPane extends JFrame
{
    public BeispielScrollPane()
    {
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setTitle("Beispiel JScrollPane");
        Container fensterinhalt = this.getContentPane();

        /* Mehrzeiliges Textfeld erzeugen und in ein scrollbares Fenster einfügen */
        JTextArea text = new JTextArea();
        text.setText("Dies ist ein langer Text im Textfeld. " + "Er passt nicht in die Fensterbreite" + "Daher erscheint  
ein horizontaler Rollbalken");
        text.setColumns(10);
        text.setRows(1);

        JScrollPane scrollpane = new JScrollPane(text);
        fensterinhalt.add(scrollpane);

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

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

189

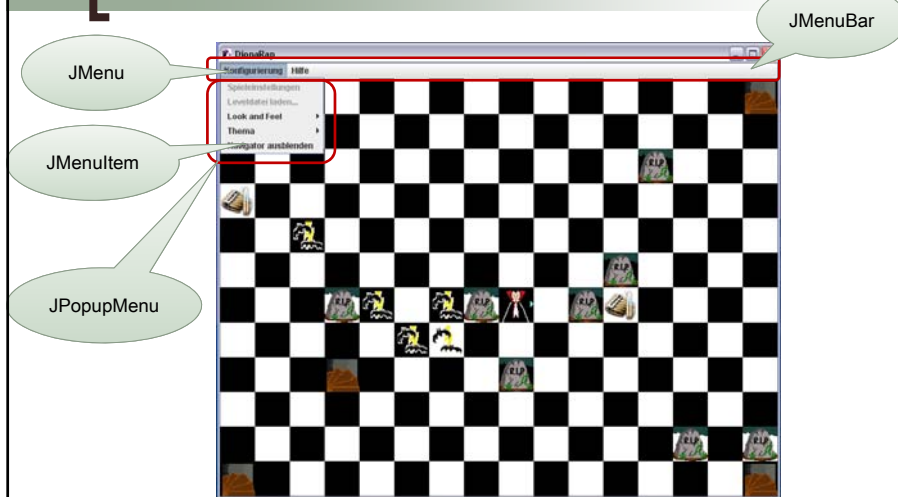
Progressbar

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

190

DionaRap erweitern mit einer Menüleiste

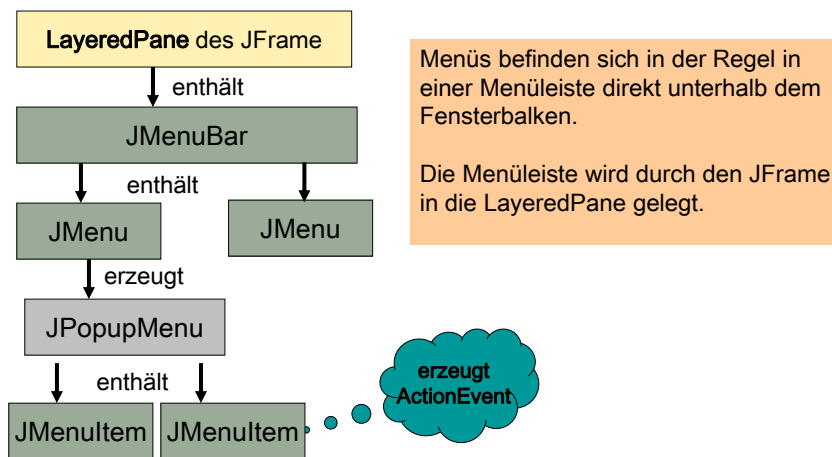


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

191

Menüs in swing



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

192

Menüs erzeugen

- ✖ 1. Schritt: JMenuBar instanziiieren und in einen JFrame einfügen
Methode: `<JFrame>.setJMenuBar(JMenuBar mb)`
- ✖ 2. Schritt: Nach Erzeugen der JMenuBar muss man Menüobjekte erzeugen und mit `<JMenuBar>.add(Menu)` in die JMenuBar einfügen
Das Menü erzeugt dann ein leeres **PopUp-Fenster** (JPopupMenu), in das die Menü-Einträge (JMenuItem) eingefügt werden können
- ✖ 3. Schritt: Erzeugen von Objekten des Typs JMenuItem und mit `<JMenu>.add(Menuitem)` in das Popup-Fenster des Menüs einfügen
Bei Anwahl eines MenuItem erzeugt dieses einen **ActionEvent**
- ✖ 4. Schritt: Zur Behandlung dieser Events muss ein **ActionListener** erzeugt und bei den JMenuItemns registriert werden.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

193

Menüs gestalten und kaskadieren

- ✖ JMenu erzeugt den Container **JPopupMenu**
- ✖ JPopupMenu ist ein Popup-Fenster und kann folgende Objekttypen beinhalten
 - JMenuItem
 - bestehen aus Text und/oder Icon
 - **RadioButtons** (JRadioButtonMenuItem), **Checkboxes** (JCheckBoxMenuItem),
 - Listen, Comboboxen ...
 - Untermenüs (JMenu)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

194

Menüs gestalten und kaskadieren

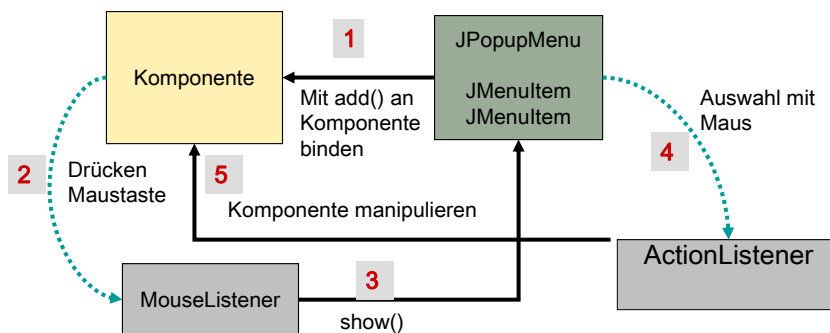
- ✖ Weiterführende Informationen findet man in einem Tutorial auf den Webseiten von Sun
- ✖ <http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

195

JPopupMenu



Beispielprogramm BeispielPopupMenu:
<http://erde.fbe.fh-weingarten.de/keller>
→ Grafische Bedienoberfläche

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

196

DionaRap MT MultiThreading

- ✖ Gegner bekommen eine Eigendynamik
 - Wird der Spieler während einer festgelegten Startzeit nicht bewegt, beginnen die Gegner sich selbständig zu bewegen
 - Gegner laufen in einem eigenen Thread ab
 - Bewegungsstrategie: finde den kürzesten Weg von meiner Position zur Position des Spielers (A*-Algorithmus)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

197

Multithreading in Java

- ✖ Multithreading bietet die Möglichkeit, mehrere Vorgänge gleichzeitig auszuführen
 - Aktionen innerhalb eines Programms werden entkoppelt
- ✖ Es gibt zwei Möglichkeiten einen Thread in Java zu erzeugen
 - ableiten von der Klasse Thread
 - Implementieren des Interfaces Runnable

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

198

Klasse Thread

- ✖ Java stellt die Klasse `java.lang.Thread` zu Verfügung
 - Diese implementiert "leeren" Thread ohne Funktion
- ✖ Statische Methoden der Klasse Thread sprechen immer den "laufenden Thread" an
- ✖ Statische Methoden von Thread
 - `Sleep()`
 - `Interrupted()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

199

Erzeugen und Starten von Threads

- ✖ **Erzeugen eines Threads**
 - Eigene Klasse von Thread ableiten und ein Thread-Objekte mit `new()` erzeugen
 - die `run()`-Methode in der abgeleiteten Klasse implementiert die eigentliche Thread-Aktivität
 - Die erzeugte Threadinstanz ist zunächst passiv, wartet auf einen Startschuss
- ✖ **Thread starten**
 - `start()` führt zum Aufruf von `run()`
 - `start()` darf nur 1× aufgerufen werden und kehrt *sofort zurück zum aufrufenden Thread*
 - `run()` ist eine Callback-Methode und von der JVM aufgerufen. Daher darf man die `run()`-Methode nicht direkt aufzurufen, sondern mittelbar über Methode `start()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

200

Klasse Thread

Beispiel „ticker“

```
public class Ticker extends Thread
{
    private String text;

    public Ticker(String text) // Konstruktor
    {
        this.text = text;
    }

    public void run()           // Aktionen des Thread
    {
        while(true)
        {
            System.out.print(text + "..."); // Ausgabe tickertext
        }
    }
}

Public class Main
{
    public static void main(String[] args)
    {
        Ticker t1 = new Ticker("tick"); // Threadinstanz erzeugen
        t1.start();                     // Thread starten
        Ticker t2 = new Ticker("tack");
        t2.start();
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

201

Thread beenden

× Freiwilliges Ende

→ Thread endet mit Ende von run()

× Unfreiwilliges Ende

- ein Thread A kann mit Methode `interrupt()` einem Thread B signalisieren, das dieser sich beenden soll
- Die Methode `interrupt()` bricht den Empfänger-Thread *nicht* ab, sondern setzt in diesem ein Flag "Bitte Beenden"
 - Der Empfänger-Thread kann dieses Flag abfragen und danach ggf. freiwillig enden
 - blockierende Methoden wie `sleep`, `wait` und `join` werfen eine `InterruptedException`. Mit `try... catch` kann man auf diese Exception reagieren und beenden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

202

× Unfreiwilliges Ende

- Abfragen des Interrupt-Flags im Thread, der beendet werden soll
 - – ...im **laufenden Threads** mit statischer Methode `interrupted()` der Klasse `Thread` (`Thread.interrupted()`)
 - diese Methode löscht das Interruptflag im laufenden Thread
 - – ...eines **anderen Threads** mit `<Threadobjekt>.isInterrupted()`
 - Diese Methode verändert das Interrupt-Flag nicht

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

203

Beispiel „ticker“ mit Interrupt

```
class Ticker extends Thread
{
    [...]
    public void run()
    {
        // while(!this.isInterrupted()) → alternative Möglichkeit einen Interrupt abzufragen

        while( !Thread.interrupted() ) // Interrupt abfragen über statische Methode
        {
            System.out.print(text + "..."); // Ausgabe nur, wenn kein Interrupt vorhanden
            System.out.flush();
        }
    }
}

class Main
{
    public static void main(String[] args)
    {
        Ticker t1 = new Ticker("tick");
        t1.start(); // Ticker 1 starten
        Thread.sleep(1000); // warte 1 Sekunde
        t1.interrupt(); // beende Ticker 1
        Ticker t2 = new Ticker("tack");
        t2.start(); // starte Ticker 2
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

204

Interface Runnable

- ✖ Eine eigene Klasse kann das Interface Runnable implementieren und damit als Thread verwendet werden
 - Das Interface Runnable deklariert nur die Methode run()
 - Man kann eine Klasse, die von einer swing-Klasse abgeleitet ist, in einem eigenen Thread laufen lassen
- ✖ Ein Unterschied liegt nur im Erzeugen des Thread-Objektes, das weitere ist Verhalten identisch
 - 1. Schritt: die eigene Klasse Instanzieren
 - man erhält eine runnable Instanz
 - Dieses Objekt ist aber noch kein Threadobjekt
 - 2. Schritt: Thread erzeugen mit
Thread t = new Thread(runnable Instanz)
 - 3. Schritt: Thread starten
t.start()

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

205

Beispiel „ticker“

```
public class Ticker implements Runnable
{
    [...]
    public void run()
    {
        while(true)
        {
            System.out.print(text + "...");
            System.out.flush();
        }
    }
}

class Main
{
    public static void main(String[] args)
    {
        Ticker tick = new Ticker("tick"); // runnable Instanz erzeugen
        Thread t = new Thread(tick);      // Threadobjekt erzeugen
        t.start();                         // Thread starten
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

206

Threadzustände

- ✖ Thread vom Zustand „rechnend“ in den Zustand „rechenbereit“ überführen
`<rechnenderThread>.yield()`
- ✖ Thread für eine gewisse Zeit blockieren
`Thread.sleep(long millisekunde)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

207

Interprozeßkommunikation

- ✖ Objekte werden geteilt von allen Threads
 - ➔ Damit ist konkurrierender Zugriff auf ein Objekt möglich
- ✖ Probleme entstehen, wenn ein Threadwechsel zwischen Abfragen und Ändern eines Objektes stattfindet
 - ➔ Lösung: **Mutual Exklusion**
 - ➔ In Java möglich durch **Monitorkonzept**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

208

Synchronisation Monitore

- ✖ Zur Realisierung von Mutual Exclusion stellt Java das Schlüsselwort **synchronized** zu Verfügung
 - Zwei Arten der Verwendung von **synchronized**
 - **Sperren von Methoden**
`synchronized Methodenkopf`
`{ //Critical section }`
 - oder **sperren von Objekten**
`synchronized(Object)`
`{ // Critical Section }`
 - **Synchronisation über Ereignisse**
 - Warten auf ein Ereignis `wait()` Signalisieren eines Ereignisses `notify()`
 - **Wait und notify dürfen nur in einem Monitor, also in synchronized Methoden / Blöcken vorkommen**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

209

Threads und swing

- ✖ **Swing ist nicht Threadsicher**
 - Was heißt threadsicher?
 - Wie kann man swing Threadsicher machen?

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

210

Was heißt threadsicher ?

- ✖ Threadsicher bedeutet, dass Softwarekomponenten gleichzeitig von verschiedenen Threads ausgeführt werden können, ohne dass diese sich gegenseitig behindern
- ✖ Swing ist nicht threadsicher

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

211

Swing threadsicher machen

- ✖ Verändern mehrere Threads nebenläufig swing-Komponenten, müssen die Threads synchronisiert werden
- ✖ Lösung: Event Dispatch Thread

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

212

Event Dispatch Thread

- ✖ Es gibt nur genau einen Event Dispatch Thread in einem swing Programm
- ✖ Der Event Dispatch Thread stellt sicher, dass der Zugriff auf die GUI-Komponente synchronisiert wird
- ✖ Das Event Handling von swing läuft immer im Event Dispatch Thread ab

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

213

Swing threadsicher machen

- ✖ Swing stellt die Methode `invokeLater()` zu Verfügung
- ✖ `InvokeLater` übergibt ein `Runnable` Objekt an den Event Dispatch Thread und kehrt sofort danach zum Haupt-Thread zurück
- ✖ In der `run()`-Methode dieses `Runnable` Objektes wird auf die GUI-Komponente zugegriffen.

```
SwingUtilities.invokeLater(new Runnable(){  
    public void run()  
    { new JButton("OK"); // GUI-Komponente, auf die mehrere  
                        // Threads zugreifen  
    });
```

- ✖ Alle Threads, die auf das gleiche GUI-Element zugreifen, müssen dies über die Methode `invokeLater` tun

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

214

Swing threadsicher machen

Folgende Bedingungen müssen beachtet werden:

1. Der Zugriff auf gemeinsam genutzte GUI-Komponenten aus mehreren Thread sollte ausschliesslich über den Event Dispatch Thread erfolgen
2. Rechenintensive Operationen sollten nicht im Event Dispatch Thread bearbeitet werden, da sonst die Anwendung zu langsam reagiert (einfriert)

Swing threadsicher machen

- ✖ Ab Java 1.6 wurde die Klasse **SwingWorker** eingeführt, die den Ablauf nebenläufiger Threads, die mit swing arbeiten, regelt
- ✖ Ein SwingWorker-Objekt verteilt Programmaktivitäten auf drei Threads
 - den aktuell laufenden Thread
 - einen Hintergrund Thread
 - den Event Dispatch Thread

Swing Worker

- ✖ SwingWorker stellt folgende Methoden zu Verfügung
 - **doInBackground**
 - Berechnet ein Ergebnis. Die Berechnung wird in einem eigenen Thread ausgeführt
 - Muss von der eigenen SwingWorker-Klasse überschrieben werden
 - **publish() und process**
 - Process bekommt von der publish-Methode Daten gesendet
 - Process wird im *Event Dispatch Thread* abgearbeitet
 - Process muss von der eigenen SwingWorker-Klasse überschrieben werden
 - **publish()** wird in **doInBackground** aufgerufen
 - **done**
 - Wird vom *Event Dispatch Thread* abgearbeitet, wenn die **doInBackground** Method beendet wurde
 - muss von der eigenen SwingWorker-Klasse überschrieben werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

217

Swing Worker

- **execute()**
 - Diese Methode startet den SwingWorker
- ✖ Links zu Beispielen
 - Beispiel zur Berechnung von Primzahlen
 - <http://www.0x13.de/index.php/code-snippets/51-swingworker-tutorial.html>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

218

Swing Worker

- ✖ **Swing Worker ist eine Generics Klasse**

Class `SwingWorker<T,V>`

Type Parameters:

- T - the result type returned by this `SwingWorker`'s `doInBackground` and `get` methods
- V - the type used for carrying out intermediate results by this `SwingWorker`'s `publish` and `process` methods

Siehe hierzu

- ➔ <http://java.sun.com/javase/6/docs/api/javax/swing/SwingWorker.html>

SwingWorker

- ✖ **Generics in Java**

➔ Siehe hierzu

- <http://www.fh-wedel.de/~si/seminare/ws05/Ausarbeitung/5.generics/genjava0.htm>

Realisieren der Munitionsanzeige in der Toolbar Bilder auf eine Zeichenfläche malen

- ✖ Bilder werden in Java durch Objekte der Klasse **Image** repräsentiert
 - Java unterstützt nur die Dateiformate **GIF** (auch animierte GIF's), **JPEG** und **PNG**
- ✖ Bilder sind keine GUI-Komponenten sondern im Speicher geladene Pixeldaten eines Bildes.
 - Sie werden in der Regel aus Dateien in den Speicher geladen.
 - Ein im Speicher geladenes Bild (Objekt der Klasse **Image**) kann nur dadurch sichtbar gemacht werden, dass man das Bild in einen Container **zeichnet**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

221

Laden eines Bildes Toolkit-Objekt

- ✖ Das Laden eines Bildes aus einer Datei ist abhängig vom Betriebssystem.
 - **System-Objektklassen** ermöglichen den Zugriff auf plattformabhängige Variablen und Methoden.
 - Die Methoden werden von einer System-Objektklasse "**Toolkit**" zu Verfügung gestellt.
- ✖ Ein Toolkit-Objekt erhält man durch
 - Aufruf der Klassenmethode **Toolkit.getDefaultToolkit()**
 - oder durch Aufruf einer Methode vererbt von der Klasse **Component** (Komponentenmethode) **getToolkit()** (z.B. **<meineKomponente>.getToolkit()**)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

222

- × Ein Toolkit-Objekt stellt dann die Methode

- `Image getImage(String dateiname)`

- `Image getImage(URL url)`

zu Verfügung, mit der man ein Bild aus einer Datei laden kann.

Als Ergebnis der Methode erhält man ein Objekt der Klasse `Image`.

- × Die Toolkit-Methode `getImage(String dateiname)` verlangt als Dateiname eine Datei mit ihrem Pfadnamen (absolut oder relativ zum Projektverzeichnis).

- Den Pfadname einer Datei erhält man über die Klasse `System`.

- × Mit der Klassenmethode
`String System.getProperty(String eigenschaftsname)`
kann man Systemparametern abfragen

Laden eines Bildes

Systemeigenschaften sind u.a.:

- | | |
|-------------------|-------------------------------------|
| → java.version, | Nummer der Java-Version |
| → os.name, | Betriebssystemname |
| → os.arch, | Rechnerarchitektur |
| → file.separator | Trennzeichen für Verzeichnisse |
| → user.dir | aktuelles Verzeichnis des Benutzers |
| → user.home | Home-verzeichnis des benutzers |
| → java.class.path | eingestellter Klassenpfad |
| → user.name | Benutzername |

Beispiel:

```
String pfadname=System.getProperty("user.dir");
String trenner=System.getProperty("file.separator");
String datei = "fh.gif";
String dateiname=pfadname+trenner+datei;
Bild=bild=this.getToolkit().getImage(dateiname);
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

225

Laden eines Bildes

- × Durch die Methode `getImage()` wird der Ladevorgang nicht sofort gestartet sondern nur der Ort des zu ladenden Bildes festgelegt
 - Geladen wird das Bild erst dann, wenn es tatsächlich angezeigt werden soll oder aber Eigenschaften des Bildes wie Breite und Höhe benötigt werden
- × Geladen wird das Bild durch einen eigenen `Thread`, der parallel zur eigentlichen Anwendung abläuft.
 - Der Thread informiert ständig alle registrierten `ImageObserver` über den Ladezustand

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

226

Laden eines Bildes

- × Ein ImageObserver-Objekt implementiert dazu das **Interface Image-Observer**, welches nur die Methode `imageUpdate()` deklariert.
 - Die Objektklasse **Component** implementiert dieses Interface. Somit kann jede Komponente als Image-Observer verwendet werden.
- × Ein ImageObserver besitzt folgende Flags
 - **WIDTH** Gesetzt wenn die Breite des Bildes bekannt ist
 - **HEIGHT** Gesetzt, wenn die Höhe des Bildes bekannt ist
 - **ALLBITS** Gesetzt, wenn alle Pixel des Bildes verfügbar sind
 - **ERROR** gesetzt beim Ladevorgang ein Fehler entdeckt wurde

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

227

Anzeigen eines geladenen Bildes

- × Ein Bild (`Image`) ist selbst keine GUI-Komponente.
- × Zur Darstellung des Bildes muss das Bild über ein **Graphics-Objekt** in eine GUI-Komponente gezeichnet werden
- × Dazu stellt das Graphics-Objekt die Methode `drawImage()` zu Verfügung
 - `drawImage(Image i, int x, int y, ImageObserver o)`
 - Als ImageObserver ist hier die Komponente anzugeben, in die zu zeichnen ist
 - `drawImage(Image i, int x, int y, int width, int height, ImageObserver o)`
 - In dieser Methode wird das Bild **skaliert** und im angegebenen Rechteck dargestellt

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

228

Zeichnen in ein Fenster

- ✖ Zum Zeichnen benötigt man ein **Objekt der Klasse Graphics**.
- ✖ Die Objektklasse Graphics ermöglicht das Zeichnen von Formen und Texten in Zeichenflächen wie JWindow, JFrame und JPanel
- ✖ Graphics enthält den grafischen Kontext, mit dem eine Komponente gezeichnet wird
 - Der grafische Kontext definiert
 - die Komponente, in die zu zeichnen ist
 - die aktuelle Farbe
 - die aktuelle Schriftart
 - Farb- und Darstellungsmodus
 - weitere Clipping- und Renderinginformationen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

229

Beispiel: Zeichnen von Text in ein Fenster

- ✖ Bisher Textausgabe über Text-Komponenten
 - Label, Textfeld, Textarea
- ✖ Alternative ist das zeichnen von Text direkt ins Fenster mit
 - Gezeichnet wird in der **paint()-Methode** eines Fensters
 - In der paint-Methode erhält man ein Malobjekt der Objektklasse **Graphics**
 - Das Graphics-Objekt stellt Methoden zum zeichnen zu Verfügung wie z.B. die Methode `<graphics>.drawLine()`



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

230

Zeichnen mit der Objektklasse Graphics in ein awt-Fenster

- × **Überschreiben der Methode `paint(Graphics g)`**
 - die `paint()`-Methode wird immer dann aufgerufen, wenn das Fenster neu zu zeichnen ist, wie z. B. bei Größenveränderungen, nach Verschieben des Fensters, wenn das Fenster vom Hintergrund in den Vordergrund geholt wird oder wenn explizit über die Methode `repaint()` ein neues Zeichnen erzwungen wird.
- × Zu jeder **sichtbaren** Komponente existiert ein Graphics-Objekt. Dieses wird als Parameter in die Methode `paint()` einer Komponente übergeben.
 - Mit der Methode `<Komponente>.getGraphics()` kann man sich eine Referenz auf ein existierendes Graphicsobjekt auch außerhalb von `paint()` besorgen, jedoch nur dann, wenn die Komponente dargestellt ist. Eine nicht dargestellte Komponente liefert null, da kein gültiges Graphicsobjekt existiert.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

231

Zeichnen mit der Objektklasse Graphics Beispiel

- × **1. Schritt:**
 - Definition einer eigenen Klasse, die man von einem Fenster z.B. `JFrame` ableitet. Diese Klasse erbt die Methode `void paint(Graphics g)`
- × **2. Schritt**
 - Überschreiben der Methode durch Deklaration einer Methode `void paint(Graphics g)`.
- × **3. Schritt**
 - Malfunktionen des Graphicsobjektes in `paint()` zum zeichnen verwenden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

232

Zeichnen mit der Objektklasse Graphics Beispiel

```
public class BeispielSmiliezeichnen extends JFrame
{
    BeispielSmiliezeichnen() {
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(200,200);
        this.setVisible(true);
    }

    public void paint(Graphics g)
    { Graphics2D g2d = (Graphics2D) g;
      int x = 100;
      int y = 100;

      // Smilie
      g2d.drawOval(x-10,y-10,20,20); g2d.setColor(Color.yellow); g2d.fillOval(x-10,y-10,20,20);
      // Augen
      g2d.setColor(Color.black); g2d.fillRect(x-6,y-5,4,5); g2d.fillRect(x+3,y-4,4,5);
      // Mund
      g2d.drawArc(x-7,y-7,14,14,225,100);

    }

    public static void main(String[] args) { new BeispielSmiliezeichnen(); }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

233

Zeichnen mit der Objektklasse Graphics in Swing-Komponenten

- ✖ In Swing-Komponenten werden in der paint()-Methode drei weitere Methoden in folgender Reihenfolge aufgerufen
 - ➔ protected void paintComponent(Graphics g)
 - ➔ protected void paintBorder(Graphics g)
 - ➔ protected void paintChildren(Graphics g)
- ✖ In Swing sollte man daher die Methode paintComponent() *verwenden und nicht paint() selbst*,
 - *paintBorder() und paintChildren() würden sonst nicht mehr aufgerufen werden*

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

234

Linienstile und Linienbreite

- × Ab Java 1.2 mit der Objektklasse `Graphics2D` möglich
 - ermöglicht das Zeichnen von Linien mit verschiedenen Linienstilen und Linienbreiten, durch Verwenden von Stroke-Objekten
 - Methode: `<Graphics2D>.setStroke(Stroke)`
- × **Wie erhält man ein Graphics2D-Objekt ?**
 - Durch casting eines Graphics-Objektes

Beispiel:

```
public void paint(Graphics g)
{   Graphics2D Grafikobjekt = (Graphics2D) g; }
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

235

Linienstile und Linienbreite

- × Wie erhält man ein Stroke Objekt ?
 - Durch Implementieren des Interface `Stroke`
 - Das Interface stellt unterschiedliche Pinsel zum Zeichnen von Randlinien zu Verfügung
 - `BasicStroke` ist die einzige implementierte Objektklasse
 - `new BasicStroke()`, `new BasicStroke(float Linienstaerke)`

Beispiel:

```
/* Linienbreite auf 3 stellen */

BasicStroke Linienbreite=new BasicStroke((float)3.0);
Graphicsobjekt.setStroke(Linienbreite);
Grafikobjekt.drawLine(30,50,300,50);
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

236

Methoden der Objektklasse Graphics

- × setBackground(Color) *Hintergrundfarbe festlegen
(nur in Graphics2D)*
- × clearRect(x,y,breite,höhe) *Rechteck mit Hintergrundfarbe füllen*
- × setColor(Color) *Zeichenfarbe festlegen*
- × setFont(Font) *Schriftart für Text festlegen*
- × draw.... - Methoden *Zeichnen von Texten oder Formen*
 - Char, String
 - Arc, Oval
 - Polygon, Polyline, Line
 - Rect, RoundedRect
 - drawImage *Füllen von Flächen*
- × fill....-Methoden
 - Arc, Oval
 - Rect, RoundedRect, Polygone

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

237

Anzeigen eines geladenen Bildes

- × Ein Bild (Image) ist selbst keine Komponente. Zur Darstellung des Bildes muss das Bild über ein Graphics-Objekt in eine Komponente gezeichnet werden (siehe paint())
- × Dazu stellt das Graphics-Objekt die Methode drawImage() zu Verfügung
 - drawImage(Image i, int x, int y, ImageObserver o)
 - Als ImageObserver ist hier die Komponente anzugeben, in die zu zeichnen ist
 - drawImage(Image i, int x, int y, int width, int height, ImageObserver o)
 - In dieser Methode wird das Bild *skaliert* und im angegebenen Rechteck dargestellt

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

238

Anzeigen eines geladenen Bildes

- ✖ Bevor man ein Bild zeichnet kann man es im Hintergrund laden
 - Dazu stellt die Objektklasse Component die Methoden `prepareImage()` und `checkImage()` zu Verfügung.
 - Boolean `prepareImage(Image i, ImageObserver o)`
Boolean `prepareImage(Image i, int width, int height, ImageObserver o)`
 - Ergebnis der Methode ist ein boolean, der anzeigt ob das Bild zum zeichnen fertig ist oder nicht
 - int `checkImage(Image, Imageobserver)`
int `checkImage(Image, width, height, ImageObserver)`
 - Diese Methode liefert den Status eines vorher ausgelösten Ladevorgangs. Ergebnis ist die Bit-ODER-Verknüpfung der Flages eines ImageObservers

Achtung: checkImage startet keinen Ladevorgang sondern fragt nur die Statusbits des ImageObservers ab.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

239

Beispiel Bildbetrachter

- ✖ Aufgabe:
 - In einem Fenster sollen beliebige Fotos dargestellt werden
 - Passen die Fotos nicht in ein Fenster sind die Bilder zu skalieren
 - Die Fotos werden über einen File-Dialog ausgewählt

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

240

MediaTracker

- × Ein Objekt der Klasse **MediaTracker** ermöglicht das Laden und überwachen mehrerer Bilder. Dazu führt ein MediaTracker-Objekt eine **Beobachtungsliste**.
 - Mit **addImage()** kann mein Bild in die Liste einfügen.
- × Bilder können zu einer **Gruppe** zusammengefasst werden. Jede Gruppe erhält einen **Index** (ID). Dieser Index bestimmt auch die Priorität mit der ein Bild bzw. eine Bildgruppe geladen wird.
Kleinere Nummern bedeuten höhere Priorität
 - Mit den Methoden **checkAll()** oder **checkID()** kann man feststellen ob alle Bilder der Liste oder alle Bilder einer Gruppe schon geladen wurden
 - Mit **waitForAll()** oder **waitForID()** wartet man, bis die Bilder vollständig geladen sind. Dann erst wird gezeichnet.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

241

Beispiel MediaTracker

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

public class BeispielMediaTracker extends JFrame
{
    Image bild;

    BeispielMediaTracker()
    {
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(100,100);
        this.setVisible(true);
        bild=this.getToolkit().getImage("fh.gif");
        MediaTracker medienverwaltung= new MediaTracker(this);
        medienverwaltung.addImage(bild,0);

        try
        { medienverwaltung.waitForAll(); repaint(); }
        catch(InterruptedException e){}
    }

    public void paint(Graphics g)
    {
        g.drawImage(bild,0,30,this); /* Position 0,0 zeichnet das Bild in den
        Fensterbalken */
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

242

Alternatives Beispiel: Anzeigen von Bildern als Label

× Beispielprogramm mit Bildern als Labels

- Auswahl des Bildes über `FileChooser`
- Filter für `FileChooser`, so dass nur Jpeg-/gif Bilder angezeigt werden
- Nach Auswahl der Datei entfernen altes Label
`zeichenflaeche.remove(bildlabel);`
- Erzeugen eines neuen `ImageIcon` und Labels
`bild=new ImageIcon(bilddatei);`
`bildlabel=new JLabel(bild);`
- Hinzufügen des Labels in den Container
`zeichenflaeche.add(bildlabel);`
- Neu darstellen
`this.pack();` /* optimale Grösse einstellen */
`this.validate();` /* Container neu darstellen */

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

243

Image Bibliotheken

× JIMI (Image Management Interface)

- Ursprünglich von **Activated Intelligence**, wird das Softwarepaket jetzt von SUN zu Verfügung gestellt
<http://java.sun.com/products/jimi/>
- Unterstützt viele gängigen Bildformate wie GIF, JPEG, TIFF, PNG, PICT, Photoshop, BMP, Targa, ICO, CUR, Sunraster, XBM, XPM, PCX
- Basierend auf Java image I/O. Die Java Image I/O API ist verfügbar seit Java 2 Version 1.4.0.
(<http://java.sun.com/j2se/1.4.2/docs/guide/imageio/index.html>)
- Methoden zum Laden und Speichern von Bildern. Damit kann man einfach Bildformate konvertieren.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

244

Beispiel: Bilddatei konvertieren

```
import Jimi.src.classes.com.sun.jimi.core.Jimi;
import com.sun.jimi.core.JimiException;

Image image = Jimi.getImage("audi.jpg");           // Bild einlesen

// Bild speichern - Achtung : Für gif und tiff gibt es keine Encoder

try {
    Jimi.putImage(image, "c:/temp/test.png");
} catch (JimiException e)
{
    JOptionPane.showMessageDialog(this, "Bild kann nicht
konvertiert werden", "Fehler", JOptionPane.ERROR_MESSAGE);
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

245

Image Bibliotheken

- ✖ JAI (Java Advanced Image API)
 - Das Paket ist sehr umfangreich und nicht einfach zu handhaben. Es ist für anspruchsvolle Aufgabenstellungen in der Bildbearbeitung gedacht
 - <http://java.sun.com/javase/technologies/desktop/media/jai/>
 - Dokumentation dazu unter:
http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/JAITOC.fm.html
 - Unterstützt viele gängige Bildformate z.B. TIFF, BMP, PNG, PNM, JPEG, GIF

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

246

Bildbearbeitung in Java

× Basis bilden drei Interfaces

- ImageProducer
- ImageConsumer
- ImageObserver

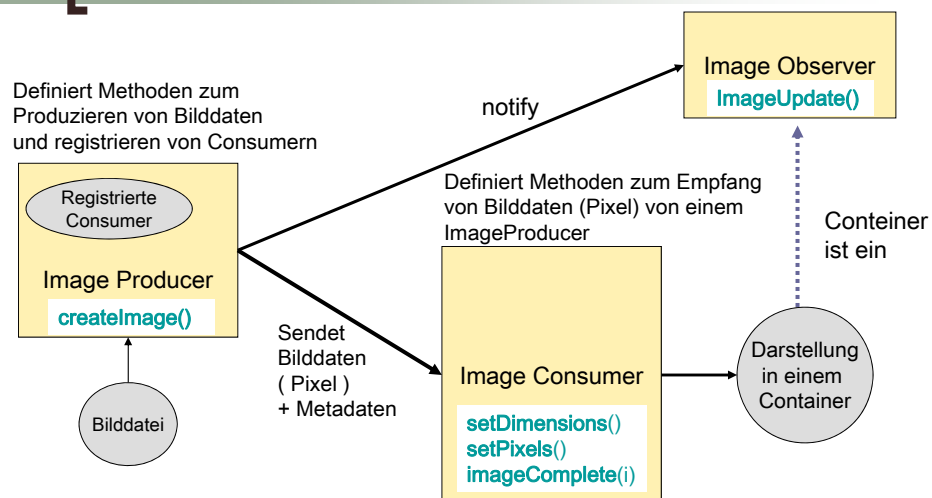
- Das Laden eines Bildes (Image) erfolgt in einem eigenen Thread, so dass das Programm ohne Blockierung weiterlaufen kann.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

247

Image Producer / Consumer / Observer



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

248

Image Producer / Consumer / Observer

× ImageProducer

- bei einem Producer können mehrere Consumer angemeldet werden
- Zuständig für das Einlesen von Bilddaten aus einer Datenquelle
 - Zuerst wird an den Consumer die Bildgröße übergeben → `setDimension()`
 - danach wird das Farbmodell geschickt
 - danach werden die Bilddaten übergeben → `setPixels()`
 - am Ende wird dem Consumer das Ende der Bilddatenübertragung gemeldet → `imageComplete()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

249

ImageProducer

- × Mit der Methode `Image.getSource()` erhält man ein Objekt vom Typ `ImageProducer`.
 - Dieses Objekt ist für die Erzeugung der Pixel im Speicher verantwortlich
 - × Die beiden Klassen
 - `MemoryImageSource`
 - Erzeugen eines Bildes im Speicher
 - `FilteredImageSource`
 - Verarbeitung/ Veränderung von Bilddaten im Speicher (Filter)
- sind `ImageProducer`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

250

Image Producer / Consumer / Observer

- ✖ ImageConsumer
 - Stellt dem Producer alle Methoden zu Verfügung, um Bilddaten empfangen zu können
- ✖ Die Klasse PixelGrabber ist ein ImageConsumer
 - Ein Objekt dieser Klasse holt sich einen rechteckigen Ausschnitt eines Image und speichert es in einem Pixel-Array

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

251

Image Producer / Consumer / Observer

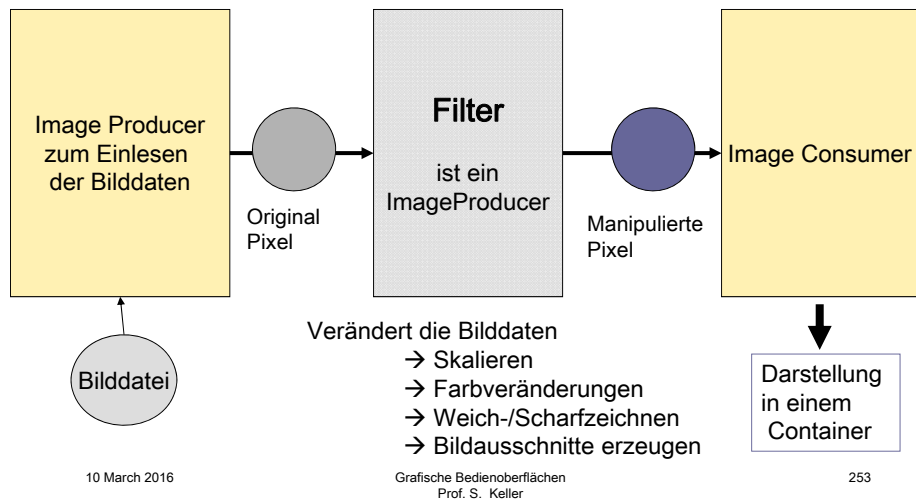
- ✖ ImageObserver wird vom ImageProducer über den Ladezustand des Bildes benachrichtigt
- ✖ Alle awt/swing-Komponenten implementieren dieses Interface
- ✖ Ein ImageObserver ist normalerweise das Objekt, in dem ein Bild gezeichnet wird

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

252

Filter für Bilddaten



Basisklasse für Filter FilteredImageSource

- × implementiert die Schnittstelle **ImageProducer**
- × **Konstruktor**
 - `FilteredImageSource(ImageProducer origImage, ImageFilter filter)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

254

FilteredImageSource

× Unterklassen

- CropImageFilter.
 - Bildteile werden herausgeschnitten.
- ReplicateScaleFilter.
 - Zum Vergrößern oder Verkleinern von Bildern.
- RGBImageFilter.
 - Zum verändern von Farben
 - Dieser allgemeine Filter ist für eigene FarbFilter-Klassen gedacht.
 - Es muss lediglich eine **filterRGB()-Methode** programmiert werden, die die RGB-Bildinformationen für jeden Punkt (x,y) modifiziert.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

255

Beispiel: FilteredImageSource

```
Image bild = getImage( "gatesInAlbuquerque.jpg" );

ImageFilter colorfilter = new GrayFilter();

ImageProducer imageprod = new FilteredImageSource( bild.getSource(),
                                                    colorfilter );

Image neuesbild = createImage( imageprod );
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

256

Weitere Themen, die in DionaRap nicht verwendet werden

- ✖ DionaRap um Soundeffekte erweitern
- ✖ Vorgefertigte Dialoge zum öffnen und speichern von Dateien
- ✖ Plugable Look&Feel
- ✖ MVC-Unterstützung in swing
 - Listen, Tabellen, Bäume
- ✖ JavaWebStart
 - Eine Java-Anwendung ins Web stellen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

257

Swing- Dialoge Vorgefertigte Dialoge

- ✖ Ein **File-Dialog** zum öffnen von Dateien wird in swing als Klasse **JFileChooser**, abgeleitet von JComponent und nicht abgeleitet von JDialog, realisiert
- ✖ Zur Auswahl von Farben kann die Klasse **JColorChooser** verwendet werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

258

Klasse JFileChooser

- ✖ Abgeleitet von JComponent
- ✖ Bei der Erzeugung zeigt das FileChooser-Objekt voreingestellt den Inhalt des **Home-Verzeichnisses** an.
 - ➔ Alternativ kann man bei der Erzeugung ein gewähltes **Startverzeichnis** angeben.
 - Das FileChooser-Objekt zeigt den Inhalt des home-Verzeichnisses an
`new JFileChooser()`
 - Der FileChooser zeigt den Inhalt des angegebenen Verzeichnisses an.
`new JFileChooser(File aktuellesVerzeichnis)`
`new JFileChooser(String PfadAktuellesVerzeichnis)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

259

Klasse JFileChooser

- ✖ Nach seiner Erzeugung wird das FileChooser-Objekt nicht sofort dargestellt. Es muss erst in einen **Container eingefügt** und damit sichtbar gemacht werden.
 - ➔ In den meisten Fällen ist es jedoch sinnvoll den FileChooser in einem separaten **Popup-Fenster** darzustellen

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

260

JFileChooser und Popup-Fenster

- ✖ Als Popup-Fenster eignet sich eine **modale Dialogbox**
 - Die Klasse JFileChooser stellt drei Methoden zu Verfügung, um ein erzeugtes Objekt in einem Popup-Fenster anzuzeigen.

int showDialog(Component Vater, String AuswahlTasteText)
Text der Auswahl taste frei gewählt

int showOpenDialog(Component Vater)
Auswahl taste ist mit open beschriftet

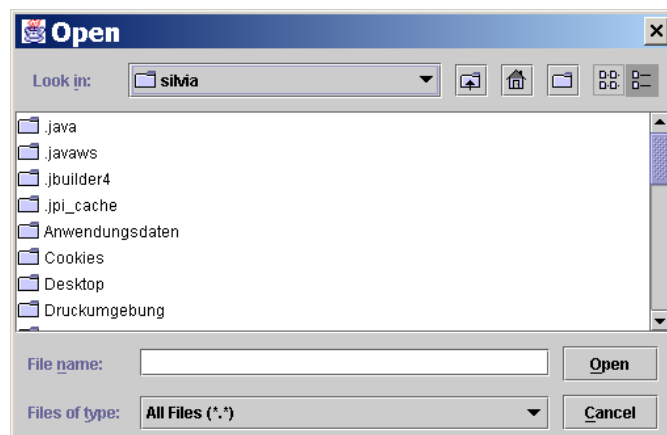
int showSaveDialog(Component Vater)
Auswahl taste ist mit save beschriftet

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

261

Die Komponente JFileChooser



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

262

JFileChooser und Popup-Fenster

- ✖ Das Popup-Fenster erscheint **zentriert** über der angegebenen **Vater-Komponente**.
 - ➔ Wird als Vater "null" angegeben erscheint das Fenster in der Mitte des Bildschirms.
- ✖ Nach dem Schließen des Popup-Fensters erhält man als return-Wert einen von drei möglichen Werten:
 - ➔ **JFileChooser.APPROVE_OPTION**
 - ➔ **JFileChooser.CANCEL_OPTION**
 - ➔ **JFileChooser.ERROR_OPTION**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

263

Klasse JFileChooser

✖ Events

Das FileChooser-Objekt besitzt die beiden Tasten "Auswahl" und "Cancel". Bei Auswahl einer dieser Taste wird ein **ActionEvent** erzeugt.

- ➔ Bei Auswahl der Taste "Auswahl" wird das Kommando **JFileChooser.APPROVE_SELECTION** im ActionEvent abgelegt
- ➔ Bei Auswahl der Taste "Cancel" das Kommando **JFileChooser.CANCEL_SELECTION**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

264

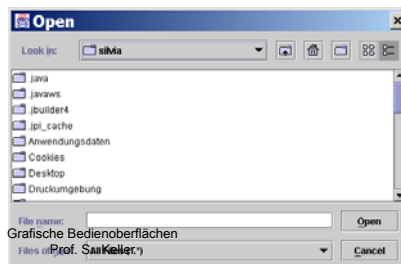
JFileChooser - Ein Beispiel

- × Klasse GUIAnwendung ist ein JFrame

```
import javax.swing.JFileChooser;
.....
JFileChooser dateiauswahl=new JFileChooser();
    dateiauswahl.addActionListener(new dateievent(this));
    dateiauswahl.showOpenDialog(this);

.....
```

Es erscheint folgende
modale Dialogbox



10 March 2016

265

JFileChooser - Ein Beispiel

Klasse dateievent implementiert einen ActionListener

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

public class dateievent implements ActionListener
{
    private GUIAnwendung anwendung;

    /* Konstruktor zur Übergabe des Anwendungs-Frame */
    public dateievent(GUIAnwendung vater) { anwendung=vater; }

    public void actionPerformed(ActionEvent e)
    {
        JFileChooser quelle=(JFileChooser) e.getSource();

        if ( e.getActionCommand().equals(JFileChooser.APPROVE_SELECTION) )
        {
            File gewaehltdatei=quelle.getSelectedFile();
            anwendung.setDateiname(gewaehltdatei.getAbsolutePath());
        }
        else System.out.println("cancel gedrueckt");
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

266

JFileChooser

× Wichtige Methoden der Klasse:

- void addActionListener(ActionListenerobjekt)
- File getSelectedFile()
- void setSelectedFile(File file)
- File[] getSelectedFiles()
- void setSelectedFiles(File[] selectedFiles)
- File getCurrentDirectory()
- void setCurrentDirectory(File dir)
- void changeToParentDirectory()

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

267

JFileChooser - File Filter

- × Sollen nur bestimmte Dateien im File Dialog angezeigt werden, muss man mit **FileFilter** arbeiten
 - FileFilter ist eine abstrakte Klasse im package **javax.swing.JFileChooser** mit zwei Methoden:
 - Mit der Methode **String getDescription()** erhält man eine Textbeschreibung, welche Dateien der Filter anzeigt
 - Mit **boolean accept(File f)** wird die Auswahl der Dateien realisiert. Eine akzeptierte Datei liefert den Wert true, eine abgelehnte Datei den Wert false

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

268

JFileChooser - File Filter

- x Um mit File Filter zu arbeiten, muss man
 - Schritt 1
 - eine eigene Klasse definieren, diese von `FileFilter` ableiten und die Methoden `getDescription()` und `accept(File f)` implementieren
 - Schritt 2
 - Der File Filter muss erzeugt werden
 - Schritt 3
 - Der `FileFilter` muss dem `JFileChooser` mit der Methode `<JFileChooser>.addChoosableFileFilter(FileFilter)` hinzugefügt werden.
 - Der voreingestellte File Filter, der alle Dateien mit `*.*` akzeptiert bleibt erhalten.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

269

File Filter – Eigene Klasse `ExtensionFileFilter`

```
public class ExtensionFileFilter extends FileFilter
{
    String beschreibung;
    String dateiextension[];

    public ExtensionFileFilter(String Beschreibung, String dateiendung[] )
    {
        beschreibung=Beschreibung;
        dateiextension=dateiendung;
    }

    public boolean accept(File f)
    {
        if ( f.isDirectory()) return true;          /* Directories acceptieren */
        else
        {
            String dateiname= f.getAbsolutePath();
            for ( int i=0; i < dateiextension.length; i++)
            {
                String endung=dateiextension[i];
                if ( dateiname.endsWith(endung) ) return true; /* Endung akzeptieren */
            }
        }
        return false;                               /* alles andere nicht akzeptieren */
    }

    public String getDescription(){return beschreibung;}
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

270

File Chooser – Eigene Klasse dateiauswahl

```
import javax.swing.JFileChooser;
import javax.swing.filechooser.*;

public class dateiauswahl extends JFileChooser
{
    dateiauswahl()
    {
        /* File Filter erzeugen */
        FileFilter jpg=new ExtensionFileFilter("jpeg", new
String[]{"jpg","JPG","jpeg","JPEG"});
        FileFilter gif=new ExtensionFileFilter("gif", new String
[]{"gif","GIF"});

        /* File Filter dem FileChooser hinzufügen */
        this.addChoosableFileFilter(gif);
        this.addChoosableFileFilter(jpg);
        this.show();
    }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

271

JFileChooser - File Filter

- ✖ Ein Beispielprogramm für
 - ➔ [FileChooser](#) und [Images](#) ist das Java-Programm Bilder
 - ➔ URL erde.fbe.fh-weingarten.de/keller/grabo.html
- ✖ Ein Beispielprogramm für
 - ➔ [FileChooser](#) mit [FileFilter](#) und [Labels](#) zum Anzeigen von [Bildern](#) liegt unter der gleichen URL

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

272

Plugable Look And Feel

- ✖ Klasse **UIManager** überwacht alle installierten Look&Feels
- ✖ Das Look&Feel kann über Klassen-Methoden der Objektklasse UIManager abgefragt und festgelegt werden
 - ➔ Das LookAndFeel ist normal auf **Metal**, einer Java eigenen Oberfläche eingestellt,

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

273

Plugable Look And Feel

- ✖ Die Klasse UIManager enthält für jedes installierte Look&Feel ein **Objekt** des Typs **LookAndFeelInfo**.
 - ➔ LookAndFeelInfo ist eine innere Klasse von UIManager
 - ➔ Ein LookAndFeelInfo-Objekt enthält eine textuelle Beschreibung des Look&Feel
 - ➔ Für jedes installierte Look&Feel wird im UIManager ein Objekte in einem array des Typs **LookAndFeelInfo[]** gespeichert

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

274

Plugable Look And Feel

- ✖ Information über installierte Look&Feel besorgen

→ `UIManager.LookAndFeelInfo[]`
`UIManager.getInstalledLookAndFeels()`

Ergebnis der Methode ist ein array von Objekten vom Typ `LookAndFeelInfo`.

- Jedes `LookAndFeelInfo`-Objekt besitzt die Methoden

- `String getName()` liefert den logischen Namen des Look&Feel
- `String getClassName()` liefert den Klassen-Namen des Look&Feel

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

275

Plugable Look And Feel Ändern des Look&Feel

- `static void UIManager.setLookAndFeel(String Klassennamen)`

Die Methode erzeugt Ausnahmen und muss daher in einem try/catch-Block aufgerufen werden.

- **Ausnahmen:**

`ClassNotFoundException:`
`InstantiationException:`

Klasse wurde nicht gefunden
Es konnte keine Instanz der Klasse erzeugt werden

`IllegalAccessException:`

Zugriff auf Klasse verweigert

`UnsupportedLookAndFeelException:`

Nicht unterstütztes Look&Feel

- **Nach der Änderung des Look&Feel müssen alle Komponenten neu dargestellt werden.** Dazu verwendet man die Klasse `SwingUtilities`.

Diese Klasse stellt die Klassenmethode

`void updateComponentTreeUI(Component c)` zu Verfügung

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

276

Plugable Look And Feel

Ändern des Look&Feel

- Beispiel für die Umstellung auf Windows
(Klassenname ist bekannt)

```
try
{
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    SwingUtilities.updateComponentTreeUI(this);
} catch (Exception e)
{ JOptionPane.showMessageDialog(this,"Fehler beim Look and feel", "MessageBox", JOptionPane.ERROR_MESSAGE);
};
```

Klassenname
für Windows

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

277

Plugable Look And Feel

Ändern des Look&Feel

- * Beispiel für die Umstellung auf Windows (Klassenname ist nicht bekannt)

```
UIManager.LookAndFeelInfo lookandfeelnamen[];
int anzahlLF;
String LFname[]=new String[10];
String LFKlasse[]=new String[10];
lookandfeelnamen=UIManager.getInstalledLookAndFeels();
anzahlLF= lookandfeelnamen.length;
for( int i=0; i < anzahlLF; i++)
{
    LFname[i]= lookandfeelnamen[i].getName();
    LFKlasse[i]=lookandfeelnamen[i].getClassName();
}
try { UIManager.setLookAndFeel(LFKlasse[1]);
    SwingUtilities.updateComponentTreeUI(this);
}
catch (Exception e)
{ JOptionPane.showMessageDialog(this,"Fehler beim Look and feel",
"MessageBox", JOptionPane.ERROR_MESSAGE); };
```

Klassenname
ermitteln

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

278

Skin Look And Feel

- ✖ Abkürzung: SkinLF
- ✖ OpenSourceProjekt
 - <http://dev.l2fprod.com/>
 - <http://www.l2fprod.com/>
- ✖ Ermöglicht in swing „Themepacks“ zur Definition eines „Look And Feel“ zu verwenden
- ✖ verwendet
 - GTK-Skins (Gimp Toolkit) UND KDE-Skins (Linux)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

279

Was ist ein Theme ?

*„Der Begriff **Theme** bezeichnet im Bereich Computer ein veränderbares Design für GUIs, siehe Skin (Computer)“*

Quelle: wikipedia

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

280

Was ist ein Skin ?

*„Ein **Skin** (engl. Haut, Verkleidung), auch **Design** oder **Theme** genannt, ist ein Paket von Bildern und Einstellungen, die das Aussehen und Verhalten der grafischen Benutzeroberfläche (GUI) von Computerprogrammen festlegen.“*

Quelle: Wikipedia

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

281

Was ist ein Skin ?

✖ Technologisch besteht ein Skin aus:

- Dateiverzeichnis, indem die Bilder der widges und Fenster als gif oder png Datei liegen
- Spezielle Datei, in der Informationen über die Verwendung und das Verhalten der Bilder beschrieben ist

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

282

Was ist ein Theme ?

- ✖ Widgets, d.h. grafische Elemente werden durch Bilder dargestellt
- ✖ Im Themepack ist beschrieben, welches Bild für welche grafische Komponente verwendet wird und wie es sich bei Skalierung und Ausrichtung verhält
- ✖ Ermöglicht die Darstellung irregulärer Fenster
 - Fenster ohne einen rechteckigen Fensterrand
 - können vom Entwickler frei gestaltet werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

283

Wie definiert man einen Skin ?

- ✖ Man erzeugt ein Verzeichnis, indem die Bilder der widgets als gif oder png Datei liegen
- ✖ Man erzeugt eine Datei, in der Informationen über die Verwendung und das Verhalten der Bilder beschrieben ist

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

284

Aufbau eines Themepack

- ✖ Wird als Zip-Datei zu Verfügung gestellt
- ✖ Das zip-Archiv enthält
 - zwei Ordner
 - In einem muss der GTK-Skin
 - im anderen der KDE-Skin liegen
 - Eine xml-Datei `skinIf-themepack.xml`
 - Diese Datei enthält Informationen über das konkrete Themepack

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

285

Gtk Skin

- ✖ wird durch die Datei `gtkrc` beschrieben
- ✖ Beschreibt das optische Verhalten der Kontrollelemente, nicht aber die Eigenschaften eines Fensters
- ✖ Enthält die Bilder für die Kontrollelemente

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

286

Kde Skin

- × Datei `kde.themerc`
- × Beschreibt das Aussehen und Verhalten von Fenstern

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

287

Verwendung in einer Java Anwendung

- × In das Projekt muss die Datei `SkinLF.jar` als externes Archiv eingebunden werden
 - Die Datei `SkinLF.jar` erhält man unter dem Link:
<http://www.l2fprod.com/download>
- × Das Themepack muss entweder als zip-Archiv oder entpackt unter das Projektverzeichnis kopiert werden
 - Themepacks kann man sich unter dem folgenden Link herunterladen:
<http://www.javootoo.com/>
- × In der main-Funktion muss angegeben werden
 - welches Themepack zu verwenden ist
 - danach muss das LookAnd Feel umgestellt oder installiert werden.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

288

Beispiele in Java

```
import com.l2fprod.gui.plaf.skin.*;
.....

public static void main(String args[]) throws Exception
/* Exception wird für Skin Look And Feel benötigt */
{
    /* Skin Look And Feel Laden und setzen */

    SkinLookAndFeel.setSkin(SkinLookAndFeel.loadThemePackDefinition(new File("coronaHthemepack\\skinlf-themepack.xml").toURL()));

    /* Skin Look And Feel installieren */
    UIManager.installLookAndFeel("coronaH", "com.l2fprod.gui.plaf.skin.SkinLookAndFeel");

    /* Skin Look and Feel als Voreinstellung setzen */
    UIManager.setLookAndFeel("com.l2fprod.gui.plaf.skin.SkinLookAndFeel");
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

289

Ein weiteres Beispiel

```
import com.l2fprod.gui.plaf.skin.*;
.....

public static void main(String args[]) throws Exception
/* Exception wird für Skin Look And Feel benötigt */
{
    Skin theSkinToUse =
        SkinLookAndFeel.loadThemePack("meinthemepack.zip");

    SkinLookAndFeel.setSkin(theSkinToUse);

    UIManager.setLookAndFeel(new SkinLookAndFeel());

    .
    .
    .
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

290

Wo finde ich Themepacks, die ich benutzen kann ?

✖ Themepacks zum herunterladen findet man unter:

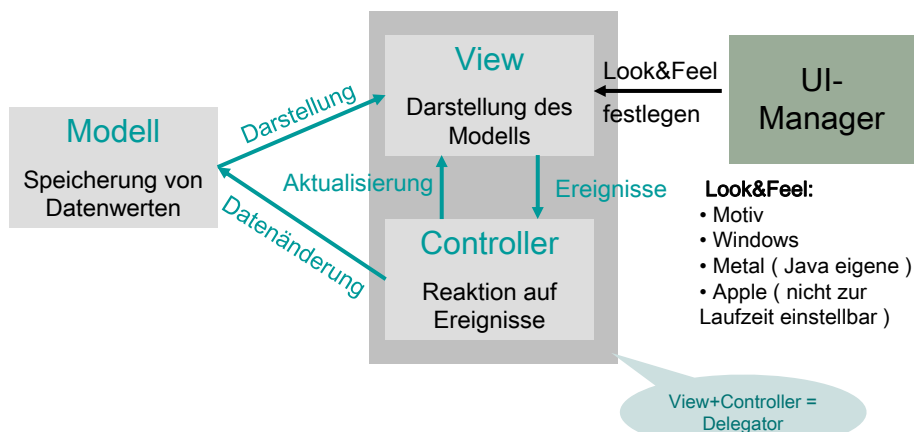
→ <http://www.javootoo.com/>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

291

MVC Programmiermodell
Eine Tabelle für die Bestenliste verwenden

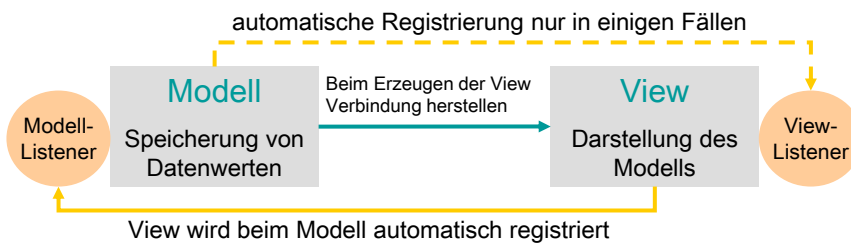


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

292

Swing Modelle und Views



- × Eine vom Modell wird mit einer View bei der Erzeugung der View verbunden (Parameter des Konstruktors)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

293

Swing – Modelle und Views

- × Swing stellt vordefinierte **Datenmodelle** als **Interfaces** zu Verfügung
- × Zur Vereinfachung stellt Java vordefinierte Klassen als Implementierung der Modelle zu Verfügung
 - Der Programmierer kann sich jedoch auch seine eigenen Modelle als Klassen implementieren
- × Zu jedem Model gehört eine passende View

Model - Interface	Vordefinierte Klasse	Passende View
ListModel	DefaultListModel	JList
TableModel	DefaultTableModel	JTable
TreeModel	DefaultTreeModel	JTree

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

294

Eine Tabelle für die Settings verwenden Das Interface `TableModel`

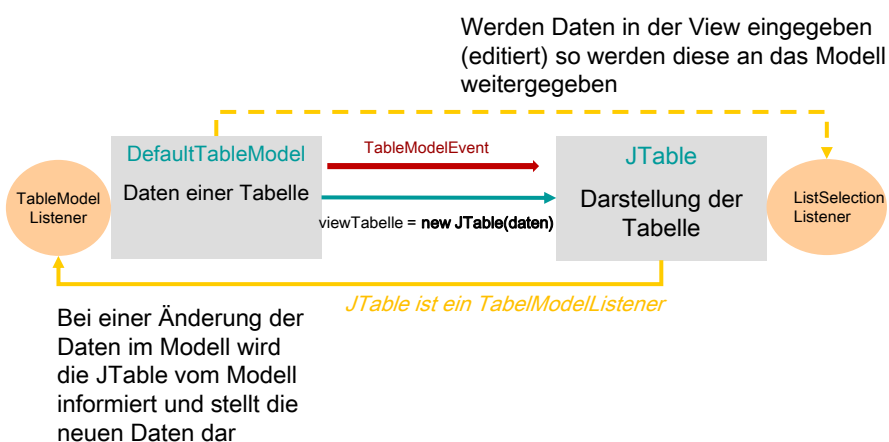
- ✖ **TableModel** deklariert folgende Methoden:
 - `getRowCount()`, `getColumnCount()`
 - `getValueAt(int rowIndex, int columnIndex)`,
 - `setValueAt(Object aValue, int rowIndex, int columnIndex)`
- ✖ Die Klasse **DefaultTableModel** implementiert das **TableModel**

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

295

Eine Tabelle für die Settings verwenden



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

296

Events bei Änderungen im Datenmodell

- × Werden im Modell Dateninhalte verändert feuert das Modell einen `TableModelEvent`
- × Gesendet werden die Events an alle registrierten `TableModelListener`
 - Einzige Methode:
 - `public void tableChanged(TableModelEvent event)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

297

Klasse `DefaultTableModel`

- × Tabelle besteht aus einem Header. Im Header werden den Spalten Namen zugeordnet
 - Methode: `this.setColumnIdentifiers(String[] header)`

```
String[] header = {"ID", "Vorname", "nachname", "kommentar"};
this.setColumnIdentifiers(header);
```
- × Daten können als ganze Zeile, als ganze Spalte
- × oder als einzelne Datenwerte[zeilenindex][Spaltenindex] bearbeitet werden
 - Methoden: `setValueAt(Objekt, zeile, spalte);`
 - `getValueAt(zeile, spalte)`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

298

Events bei Anwahl einer JTablezelle (MouseListener)

- ✖ Interface: **ListSelectionListener**

- ➔ Methode:

- ```
public void valueChanged(ListSelectionEvent event)
```

## Weiterführende Informationen zu MVC - Tabellen

- ✖ Tutorial zu JTable

- ➔ <http://docs.oracle.com/javase/tutorial/uiswing/components/table.html>

- ✖ Programmbeispiele

- ➔ <http://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html#SimpleTableDemo>

## Swing – weitere Datenmodelle

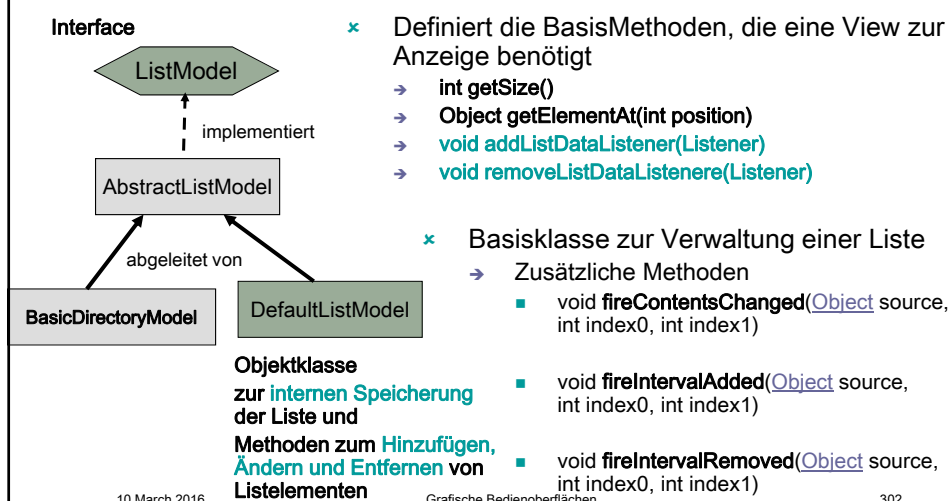
- ✖ Swing stellt nicht nur Tabellen als **Datenmodelle** zu Verfügung sondern auch Listen und Bäume
  - ➔ **Interface ListModel**
  - ➔ **Interface TreeModel**
- ✖ Zur Vereinfachung stellt Java vordefinierte Klassen als Implementierung der Modelle zu Verfügung
  - ➔ Der Programmierer kann sich jedoch auch seine eigenen Modelle als Klassen implementieren

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

301

## Swing – ListModel

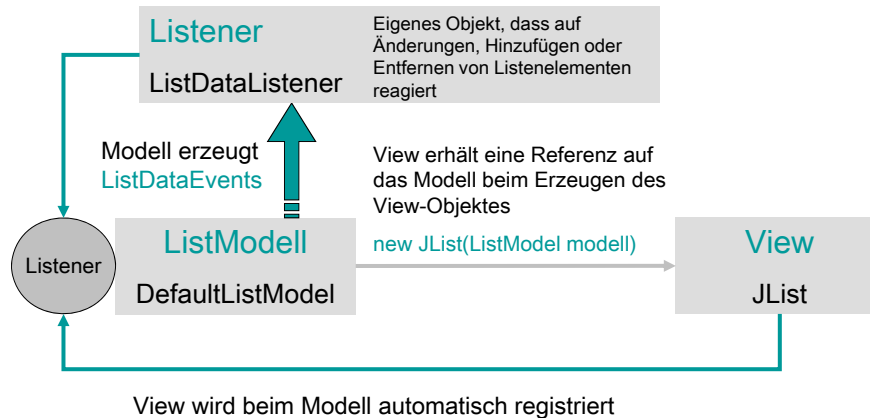


10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

302

## Swing – ListModel / JList



10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

303

## Swing – DefaultListModel Methoden

- ✖ Erzeugung eines List-Modells
  - Es gibt nur den Standard-Konstruktor  
`new DefaultListModel()`
- ✖ Eigenschaften
  - boolean empty      Methode: `isEmpty()`
  - int size              Methode: `getSize()`
- ✖ Listenerobjekt registrieren
  - `addListDataListener()`

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

304



## Swing – DefaultListModel Methoden

### → Hinzufügen von Elementen

- `add(int index, Object element)` Element an Position Index hinzufügen
- `addElementAt(Object obj, int index)`
- `addElement(Object obj)` Element ans Ende der Liste anfügen

### → Ändern von Elementen

- `Object set(int index, Object element)` Ersetzt das Element an Position index durch das angegebene Objekt. Ergebnis ist das ersetzte Objekt
- `setElementAt(Object obj, int index)` Wie set(). Nur ohne Rückgabe des ersetzten Objektes

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

305

## Swing - DefaultListModel

### → Entfernen von Elementen

- `clear()` Entfernt alle Elemente der Liste
- `removeAllElements()`
- `Object remove(int index)` Entfernt das Objekt an Position index
- `removeElementAt(int index)` Wie remove nur ohne Ergebnis
- `removeElement(Object obj)` Entfernt das erste gefundene Objekt obj in der Liste
- `removeRange(int fromIndex, int toIndex)` Entfernt alle Objekte im Bereich zwischen from Index und toindex

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

306

## Swing - DefaultListModel

### × Abfragen der Liste

- `firstElement()`                      Erstes Element
- `lastElement()`                      Letztes Element
- `getSize()`
- `isEmpty()`
- `Object get(int index)`              Objekt an Position index
- `int indexOf(Object elem)`            Suche nach einem Objekt
- `indexOf(Object elem, int index)`    Suche ab einer Position
- .....

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

307

## Swing - ListDataListener

- × DefaultListModel erzeugt bei jeder Änderung in der Liste ein `ListDataEvent`

- × `ListDataEvent` hat drei abfragbare Eigenschaftentypen

- Einen Eventtyp    Methode: `getType()`              Mögliche Werte:  
                                                                                                 `CONTENTS_CHANGED`  
                                                                                                 `INTERVAL_ADDED`  
                                                                                                 `INTERVAL_REMOVED`
- `Index0`              Methode: `getIndex0()`
- `Index1`              Methode: `getIndex1()`              den Bereich in der Liste

Index0 und Index1 geben den Bereich der Liste an, in dem Änderungen vorgenommen wurden

- × Sender des Event erfragen, also das Model-Objekt,
  - Methode `getSource()`

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

308

## Swing – ListModel Beispiel

```
import javax.swing.*;
....

/* Liste als swing ListModel erzeugen */
DefaultListModel zaehlerliste=new DefaultListModel();

/* view zur Liste erstellen und das Modell anmelden */
JList viewlist = new JList(zaehlerliste);

/* einen eigenen Listener für Änderungen im Modell erzeugen und
registrieren */
zaehlerliste.addListDataListener(new DataListener());
.....
/* View einem Container z.B. JFrame hinzufügen */
Container content = this.getContentPane();
content.add(viewlist);
```

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

309

## Swing – ListModel Beispiel

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

public class DataListener implements ListDataListener
{

 public void contentsChanged(ListDataEvent e)
 {System.out.println("Changed "+e.getIndex0()); }

 public void intervalAdded(ListDataEvent e)
 {System.out.println("Added "+e.getIndex0());}

 public void intervalRemoved(ListDataEvent e)
 {System.out.println("Removed "+e.getIndex0());}

}
```

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

310

## MVC – Datenstruktur Bäume

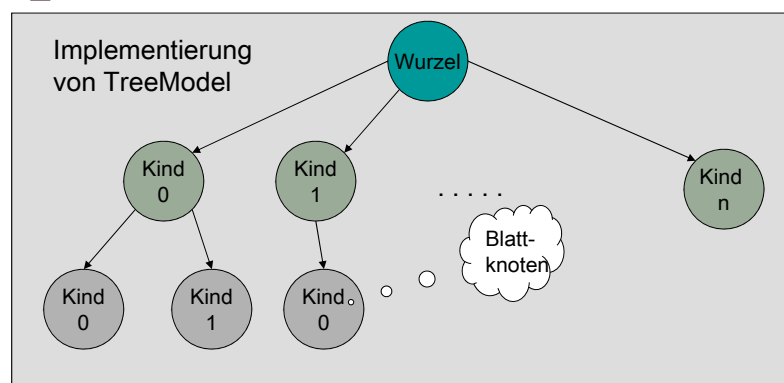
- ✖ Die Datenstruktur **Baum** wird in Java durch ein **TreeModel** repräsentiert
- ✖ Ein TreeModel enthält die Knoten des Baumes sowie alle Vater/Sohn Relationen (*Kanten zwischen den Knoten*)
- ✖ Ein Baum ist eine **dynamische Datenstruktur**. Über Methoden können Knoten hinzugefügt und entfernt werden
- ✖ Jede Änderung im Baum wird als **Event** einem **Listenerobjekt** gemeldet.

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

311

## MVC - TreeModel



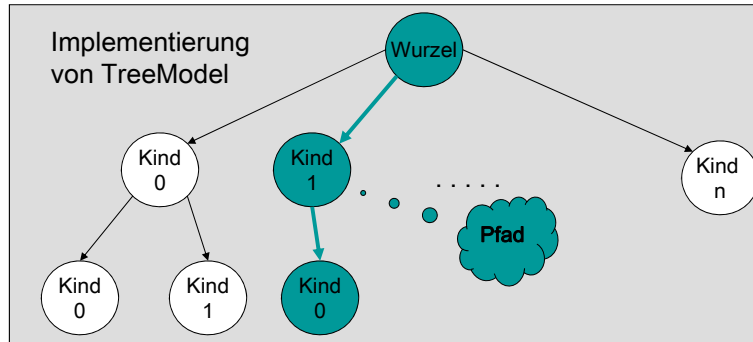
Jeder Knoten ist die Implementierung eines **TreeNode**s

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

312

## MVC - TreePath



Ein **TreePath** ist ein Objekt, welches alle Knoten des Baumes entlang eines Pfades in einem array speichert.

1. Element des array ist die Wurzel; Letztes Element ist der Endknoten des Pfades ( *im Beispiel der Blattknoten* )

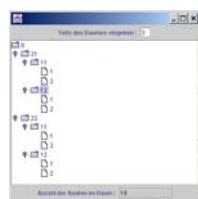
10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

313

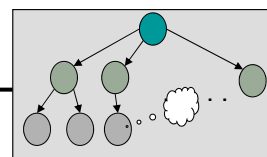
## MVC – grafische Darstellung mit JTree

View



Klasse  
**JTree**

Modell



Klasse  
**DefaultTreeModel**

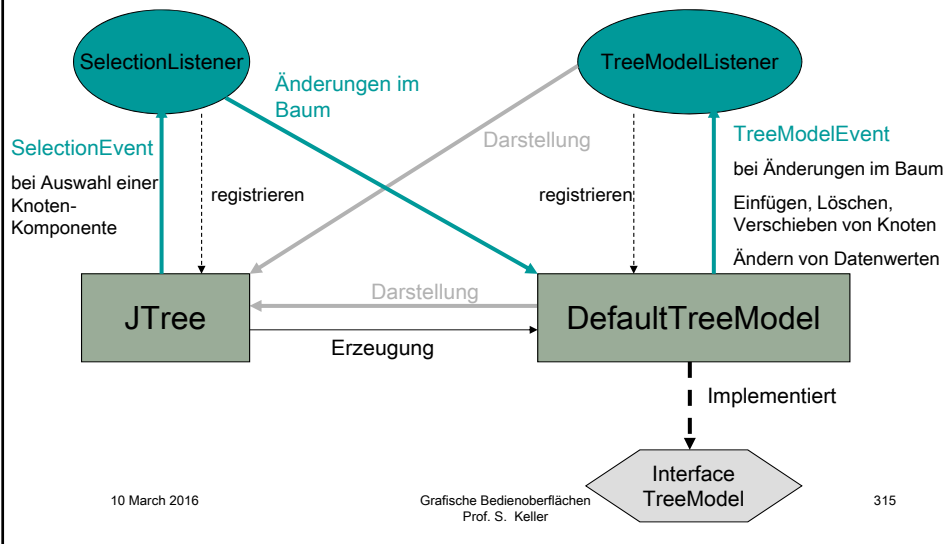
Anmeldung  
zur Darstellung

10 March 2016

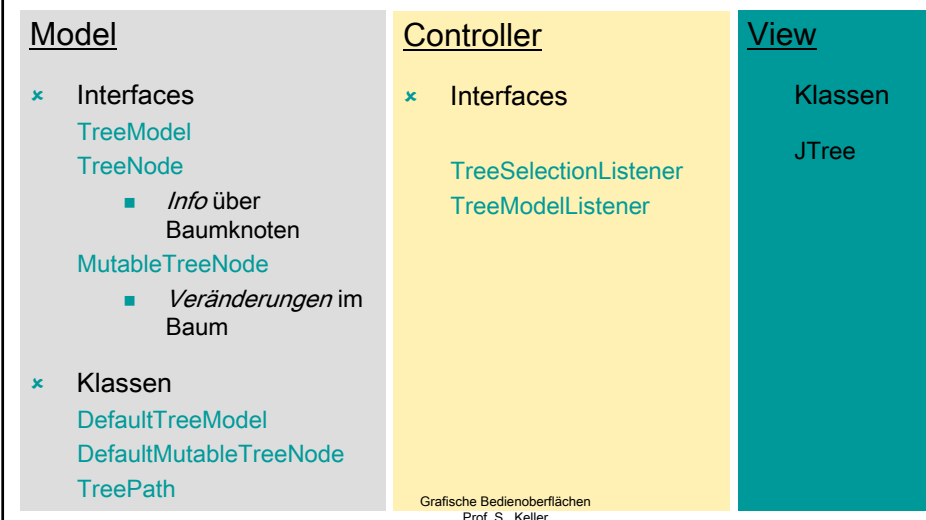
Grafische Bedienoberflächen  
Prof. S. Keller

314

## MVC - Bäume



## MVC - TreeModel



## MVC - TreeModel

### Interfaces

TreeModel  
TreeNode



implementiert

MutableTreeNode

### Klassen

DefaultTreeModel

implementiert

DefaultMutableTreeNode

#### Info über Baumnoten:

getChildCount()  
isLeaf(), isRoot()  
getParent()  
children(), getChildAt()

#### Änderungen der Baumknoten:

setParent()  
insert(), remove()

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

317

## Klasse DefaultMutableTreeNode

- ✖ Package: `javax.swing.tree`
- ✖ Eine Instanz hat maximal einen Vater und 0-n Kinder
  - ➔ Ein Instanz ohne Vater ist die Wurzel eines Baumes
  - ➔ Eine Instanz ohne Kinder ist ein Blattknoten
  - ➔ Ein Knoten mit Kindern bildet die Wurzel für einen Unterbaum
- ✖ Eine Instanz verweist auf ein Objekt, in dem die eigentlichen Benutzerdaten gespeichert sind

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

318

## Klasse DefaultMutableTreeNode

### ✖ Konstruktoren

- DefaultMutableTreeNode()
  - Erzeugt einen Knoten ohne Vater, ohne Kinder und ohne Inhalt
- DefaultMutableTreeNode( Object daten)
  - Erzeugt einen Knoten ohne Vater, ohne Kinder und eine Referenz auf den Dateninhalt
- DefaultMutableTreeNode( Object daten, boolean erlaubtKinder)

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

319

## Klasse DefaultMutableTreeNode Methoden

- **<knoten>.add(MutableTreeNode Kindknoten)**
  - Hängt Kindknoten als letztes Kind an knoten an. knoten wird damit der Vater von Kindknoten
- **<knoten>.insert(MutableTreeNode Kindknoten, int Position)**
  - hängt Kindknoten als Kind an die angegebene Position von knoten an. knoten wird damit der Vater von Kindknoten.  
Der Kindknoten darf kein Vorfahre von Knoten sein.
- **<knoten>.remove(int Position)**
  - Entfernt einen Kindknoten an der angegebenen Position. Der entfernte Knoten hat damit keinen Vater mehr
- **<knoten>.remove(MutableTreeNode Kindknoten)**
  - Entfernt den angegebenen Kindknoten. Der Kindknoten hat danach keinen Vater mehr.
- **<knoten>.removeAllChildren()**
  - Entfernt alle Kinder von knoten.

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

320



## Klasse DefaultMutableTreeNode Methoden

- **<knoten>.removeFromParent()**
  - Herauslösen eines **Unterbaumes**. Die Wurzel des Unterbaumes ist knoten.
- **<knoten>.setParent(DefaultMutableTreeNode Vaterknoten)**
  - knoten bekommt als neuen Vater den angegeben Vaterknoten. Dadurch wird ein **Unterbaum** an eine andere Position in einem Baum **verschoben**
- **DefaultMutableTreeNode <knoten>.getParent()**
  - liefert den direkten Vater von knoten. Hat dieser keinen Vater ist das Ergebnis null
- **DefaultMutableTreeNode[] <knoten>.getPath()**
  - Liefert eine Liste von Knoten beginnend bei der Wurzel bis zum aktuellen Knoten.  
Die Liste beschreibt einen Pfad im Baum beginnend bei der Wurzel des Baumes bis zu knoten

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

321

## Klasse DefaultMutableTreeNode Methoden

- **int getIndex(DefaultMutableTreeNode Kind)**
  - Liefert die Position des angegebenen Kindknoten Kind. Ist Kind kein Kindknoten ist das Ergebnis -1
- **int <knoten>.getLevel()**
  - Liefert die Anzahl von Kanten zwischen Wurzel und knoten.
- **DefaultMutableTreeNode <knoten>.getRoot()**
  - Liefert die Wurzel des Baumes, zu dem knoten gehört
- **DefaultMutableTreeNode <knoten>.getLastChild()**
  - Liefert den Kindknoten an letzter Position. Hat knoten keine Kinder wird ein Exception erzeugt
- **DefaultMutableTreeNode <knoten>.getFirstChild()**
  - Liefert den Kindknoten an erster Position. Hat knoten keine Kinder wird ein Exception erzeugt

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

322

## Klasse DefaultMutableTreeNode Methoden

- **int getChildCount()**
  - Liefert die Anzahl der Kinder des Knoten
- **Object getUserObject()**
  - Liefert das Datenobjekt, auf das der aktuelle Knoten verweist
- **setUserObject(Object daten )**
  - Setzt einen Verweis auf die Daten zu dem Knoten
- **String toString()**
  - Liefert einen String, der den Wert der Daten zu dem Knoten anzeigt. Das Datenobjekt muss eine Methode toString enthalten.
- **boolean isLeaf()**
  - wahr, falls Knoten ein Blatt ist, sonst falsch
- **boolean isRoot()**
  - wahr, wenn Knoten die Wurzel ist, sonst falsch
- **boolean isNodeChild(TreeNode Kind)**
  - liefert wahr, wenn Kind ein Kindknoten vom aktuellen Knoten ist, sonst falsch. Ist Kind null ( existiert also nicht ) wird falsch zurückgegeben.

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

323

## Klasse DefaultTreeModel

- × Package: javax.swing.tree
- × Diese Klasse beschreibt das Datenmodell Baum. Eine Instanz der Klasse enthält die Knoten des Baumes ( Typ: TreeNodes )
- × Einstieg in den Baum ist der Wurzelknoten. Von diesem Knoten aus können alle Kinder besucht werden.
- × Konstruktor: **DefaultTreeModel(TreeNode wurzel)**

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

324

## Klasse DefaultTreeModel Methoden

- addTreeModelListener(TreeModelListener)
- getRoot()
- InsertNodeInto()
- removeNodefromParent()
- nodeChanged()
- nodesWereInserted()
- nodesWereRemoved()
- getChild()
- getPathToRoot()

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

325

## Klasse TreePath

- ✖ Package: javax.swing.tree
- ✖ Diese Klasse repräsentiert den Pfad in einem Baum beginnend mit der Wurzel bis zu einem Knoten k. Die Knoten des Pfades werden in einem array gespeichert. Das erste Element im array ist die Wurzel. Das letzte Element der Knoten k.
- ✖ Arbeitet man mit einem Datenmodell TreeModel, so kann man sich eine Pfad erzeugen lassen mit der Methode
  - Baummodell.getPath(TreeNode k)

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

326

## Klasse TreePath Methoden

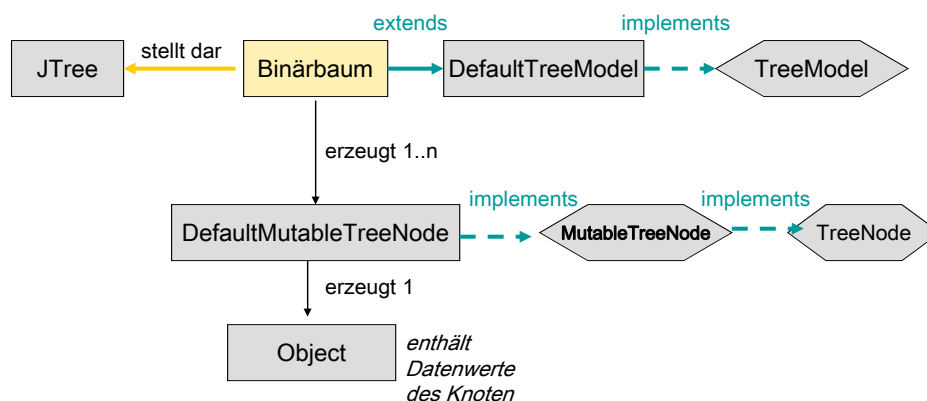
- Object getLastPathComponent()
  - Liefert das letzte Element im Pfad, also den Knoten k
- Object getPathComponent(int Position)
  - Liefert den Knoten an der angegebenen Position in der Liste
- int getPathCount()
  - Liefert die Anzahl von Knoten im Pfad
- String toString()
  - Beschreibt den Pfad als String. Dabei wird jeder Knoten als String dargestellt z.B. [0,1,11,22] wenn die Knoten Integerwerte enthalten, 0 der Wert der Wurzel ist und 22 der Wert des letzten Knoten im Pfad.

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

327

## Beispiel – voller Binärbaum



10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

328

## Beispiel – voller Binärbaum

```
import javax.swing.tree.*;

public class binaerbaum extends DefaultTreeModel
{
 /* Eigenschaften */
 DefaultMutableTreeNode wurzel;
 DefaultMutableTreeNode aktuellerknoten;
 int bbtiefe, knotenzahl;

 /* Konstruktor erzeugt einen Knoten mit Wert 0, die Wurzel des Baumes */
 public binaerbaum(int tiefe, DefaultMutableTreeNode root)
 {
 super(root);
 bbtiefe=tiefe;
 knotenzahl=1;
 aktuellerknoten=wurzel;
 wurzel=root;
 if (tiefe > 0) bauebaum(tiefe,wurzel);
 }
}
```

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

329

## Beispiel – voller Binärbaum

```
/* Rekursive Konstruktion des Baumes mit Tiefe t */
public void bauebaum(int tiefe, DefaultMutableTreeNode wurzel)
{
 DefaultMutableTreeNode links,rechts;

 if (tiefe == 0) {}
 else if (tiefe == 1)
 {
 wurzel.insert(new DefaultMutableTreeNode(new Integer(1)),0);
 wurzel.insert(new DefaultMutableTreeNode(new Integer(2)),1);
 knotenzahl=knotenzahl+2;
 }
 else
 {
 links=new DefaultMutableTreeNode(new Integer((tiefe-1)*10+1));
 wurzel.insert(links,0);
 knotenzahl++;
 bauebaum(tiefe-1,links);
 rechts=new DefaultMutableTreeNode(new Integer((tiefe-1)*10+2));
 wurzel.insert(rechts,1);
 knotenzahl++;
 bauebaum(tiefe-1,rechts);
 }
}
```

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

330

## Klasse JTree

- × Package: `javax.swing`
- × Die Klasse stellt alle Knoten in einem Baum dar. Der Baum, und damit dessen Knoten, sind in einem `TreeModel` gespeichert. Jeder Knoten im `TreeModel` wird als grafische Komponente angezeigt.
- × Knoten mit Kinder und Blattknoten werden unterschiedlich grafisch dargestellt ( analog Verzeichnis/Dateien)
- × Der Baum wird **dynamisch** dargestellt, d.h. es wird erst nur der Wurzelknoten angezeigt.
  - Enthält die Wurzel Kinder, so kann man die Komponente selektieren und die Kinder der nächsten Stufe werden dargestellt.

10 March 2016

Grafische Bedienoberflächen  
Prof. S. Keller

331

## Klasse JTree Darstellung mit Rollbalken

- × Enthält der Baum viele Knoten und Stufen, ist er im expandierten Zustand nicht vollständig darstellbar.
    - Man kann `JTree` in einem Panel mit Rollbalken (`JScrollPane`) darstellen.  
Passen die Baumknoten nicht mehr in das Panel, erscheinen Rollbalken und man kann den Baum horizontal und vertikal rollen.
- ```
/* JTree erzeugen und mit dem Modell verbinden */
JTree baumanzeige = new JTree(baummodell);
/* Fenster mit Rollbalken erzeugen und Jtree als
   Darstellungsinhalt angeben */
JScrollPane rollfenster= new JScrollPane(baumanzeige);
/* Rollfenster zur Darstellung in einen Container legen */
zeichenflaeche.add(rollfenster);
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

332

Klasse JTree

Konstruktoren

- `JTree(TreeModel)`
 - Stellt einen vorhandenen Baum an der Oberfläche dar. Der Baum ist in einem `TreeModel` gespeichert
- `JTree(TreeNode wurzel)`
 - `JTree` erzeugt ein `TreeModel`. Der Baum im erzeugten `TreeModel` besitzt nur einen Wurzelknoten `wurzel`.
- `JTree(Object[] Objektliste)`
- `JTree(Vector vektor)`
 - Für jedes Objekt im array `Objektliste` bzw. im Vektor wird ein Baumknoten erzeugt. Die Knoten liegen alle auf der gleichen Stufe und sind an einen virtuellen Wurzelknoten angehängt. Der Wurzelknoten wird nicht dargestellt.



- Beispielprogramm:

10 March 2016 <..\..\..\jbproject\LinearerBaum\LinearerBaum.jpx>
Grafische Bedienoberflächen
Prof. S. Keller

333

Klasse JTree

Darstellung von Knoten

- ✗ Die grafische Erscheinung der Knoten kann frei gestaltet werden
- Jeder Knoten wird durch ein Objekt dargestellt, welches das `Interface TreeCellRenderer` implementiert
 - Voreinstellung:
 - Ein Knoten wird durch die Komponente `JLabel` dargestellt. Das Label enthält eine Ikone und einen Text
 - Beispiele:  21  2
- Ein Knoten kann jedoch mit einer beliebigen Komponente dargestellt werden. Dafür muss man eine Klasse definieren, die das Interface `TreeCellRenderer` implementiert, eine Instanz dieser Klasse erzeugen und diese mit der Methode `<JTree>.setCellRendere(TreeCellRenderer)` bei `JTree` registrieren.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

334

- Die Klasse `DefaultTreeCellRenderer` implementiert das Interface und wird automatisch bei Erzeugung eines `JTree` erzeugt und beim `JTree` registriert. Die Klasse stellt einen Knoten durch ein `JLabel` dar
 - Einstellbare Eigenschaften von `DefaultTreeCellRenderer`
 - `background`
 - `closedIcon`, `leafIcon`, `OpenIcon`
 - `TextSelectionColor`, `textNonSelectionColor`

Klasse `DefaultTreeCellRenderer`

- × Will man Ikonen und Font verändern kann man sich von `JTree` die Instanz von `DefaultTreeCellRenderer` besorgen und dessen Eigenschaften verändern

`<JTree>.getCellRenderer()`

Achtung !!!!

Renderinformationen werden in einem Cache gespeichert.

Ändert sich die Grösse der Ikone muss neu gezeichnet werden, da der Zeilenabstand nicht mehr passt

Damit der Renderer die Daten nicht aus dem Cache nimmt, sondern neu berechnet, muss man ihn triggern.

Man erreicht dies dadurch, dass man die Zeilenhöhe auf einen negativen Wert setzt

Klasse DefaultTreeCellRenderer

- × Package: `javax.swing.tree`
- × Methoden der Klasse
 - void `setBackground`(Color color)
 - void `setBackgroundNonSelectionColor`(Color newColor)
 - void `setBackgroundSelectionColor`(Color newColor)
 - void `setBorderSelectionColor`(Color newColor)
 - void `setClosedIcon`(Icon newIcon)
 - void `setFont`(Font font)
 - void `setLeafIcon`(Icon newIcon)
 - void `setOpenIcon`(Icon newIcon)
 - void `setTextNonSelectionColor`(Color newColor)
 - void `setTextSelectionColor`(Color newColor)
- Beispielprogramm [..\..\..\jbproject\Mehrwegbaum\Mehrwegbaum.jpx](#)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

337

Klasse JTree - TreeSelectionListener

- × Steht die Eigenschaft `editable` von JTree auf `true`, so kann man die Knoten im Baum auswählen und verändern
- × Über ein Objekt `TreeSelectionModel`, das vom JTree erzeugt wird, kann man den Selektionsmodus, ob ein oder mehrere Knoten selektierbar sind verwalten.
 - Mit der Methode `<JTree>.getSelectionModel()` erhält man das `TreeSelectionModel` -Objekt.
 - Mit der Methode `<TreeSelectionModel -Objekt>.setSelectionMode(Modus)` kann man den Selektionsmodus ändern.
 - Die Selektionsmodi sind in der Klasse `TreeSelectionModel` als Konstanten definiert.
- × Selektiert man eine Knotenkomponente wird ein `TreeSelectionEvent` erzeugt und allen registrierten `TreeSelectionListener` gesendet.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

338

Klasse JTree - TreeSelectionListener

- × Konstanten für Selektionsmodus
 - **SINGLE_TREE_SELECTION**
 - Es kann nur ein Knoten selektiert werden.
 - Die Selektion wird in genau einem Pfad von der Wurzel zu dem selektierten Knoten gespeichert
 - **CONTIGUOUS_TREE_SELECTION**
 - Es können mehrere Knoten selektiert werden. Die Knoten sind benachbart. Die Selektion enthält damit mehrere Pfade.
 - **DISCONTIGUOUS_TREE_SELECTION**
 - Es können mehrere Knoten selektiert werden. Die Knoten müssen nicht benachbart liegen. Die Selektion enthält damit mehrere Pfade.
 - Voreinstellung

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

339

Klasse JTree - TreeSelectionListener

- × In einem **TreeSelectionListener**-Objekt wird auf den Event reagiert und die gewünschten Aktionen ausgeführt. Diese Aktionen verändern in der Regel den Baum.
 - **Durch diese Baumänderungen werden dann in Folge alle registrierten TreeModelListener aufgerufen.**
- × Ein TreeSelectionListener-Objekt muss das Interface TreeSelectionListener implementieren.
 - Dazu muss in der Objektklasse die Methode **public void valueChanged(TreeSelectionEvent e)** realisiert werden
- × Das Listenerobjekt muss danach instanziiert und beim JTree registriert werden, um die Events zu erhalten

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

340

Klasse JTree - TreeSelectionListener

× Beispiel

- Selektiert man einen Blattknoten, so soll dieser gelöscht werden.
 - Dadurch erniedrigt sich die Knotenzahl um 1 und eventuell ändert sich auch die Tiefe des Baumes
- Vorgehensweise
 - Man definiert sich eine Klasse `baumlistener`, die das Interface `TreeSelectionListener` implementiert
 - In dieser Klasse muss dann die Methode
`public void valueChanged(TreeSelectionEvent e)`
realisiert werden
- [..\..\..\jbproject\binaerbaum\binaerbaum.jpx](#)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

341

Klasse JTree - TreeSelectionListener

- × In der Methode `valueChanged()` wird überprüft, ob der selektierte Knoten ein Blattknoten ist, der gelöscht werden darf.
- × Wenn dies der Fall ist, wird der Knoten aus dem Modell gelöscht, die Knotenzahl um 1 erniedrigt, das Modell nach der Tiefe des neuen Baumes gefragt, um dann die Blattknoten und Tiefe im Frame neu anzeigen zu können
 - Dazu muss das Listenerobjekt den `JTree`, das `Baummodell` und den `Anwendungsframe` kennen
 - Den `JTree` erhält der Listener über die Methode `<Event>.getSource()`
 - Das `Baummodell` kann man sich über die Methode `<JTree>.getModel()` besorgen
 - Hat man `JTree` als Komponente erzeugt, die in der Anwendung liegt, so kann man sich über die Methode
 - `<JComponent>.getTopLevelAncestor()`
den `Anwendungsframe` besorgen, ohne die lange Kette von `getParent()` benutzen zu müssen.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

342

Klasse JTree TreeSelectionListener

```
import javax.swing.event.*;
import javax.swing.tree.*;
import javax.swing.*;
import java.util.*;

public class baumlistener implements TreeSelectionListener
{
    binaerbaumUI anwendung;
    JTree baumdarsteller;
    DefaultMutableTreeNode auswahlknoten=null;

    /* Interface implementieren */
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

343

Klasse JTree TreeSelectionListener

```
public void valueChanged(TreeSelectionEvent e)
{
    baumdarsteller = (JTree) e.getSource(); /* JTree löst den Event aus */
    /* Um die neue Knotenzahl und Baumtiefe anzuzeigen wird der Anwendungsframe
    benötigt */
    anwendung = (binaerbaumUI)baumdarsteller.getTopLevelAncestor();
    TreePath pfad=e.getPath(); /* Pfad vom seletierten Knoten besorgen */
    /* Der selektierte Knoten ist das letzte Objekt im Pfad */
    auswahlknoten=(DefaultMutableTreeNode) pfad.getLastPathComponent();
    /* Die Wurzel des Baumes darf nicht gelöscht werden */
    if (auswahlknoten!=null && !auswahlknoten.isRoot())
    {
        /* Nur die Blattknoten dürfen gelöscht werden */
        if ( auswahlknoten.isLeaf() && auswahlknoten!=null )
        {
            binaerbaum baummodell=(binaerbaum)baumdarsteller.getModel();
            baummodell.removeNodeFromParent(auswahlknoten);
            /* Vorsicht die Methode removeNodeFromParent sendet einen Event an
            diesen Listener Wird dieser dann das 2. mal ausgeführt ist
            Auswahlnoten schon gelöscht ein nochmaliges löschen löst
            der einen NullPointer-Exception aus. Daher muss vor dem Löschen geprüft
            werden ob der Auswahlnoten noch existiert */
            System.out.println("knoten wurde im Modell gelöscht");
            auswahlknoten=null; ..... } } } }
}
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

344

Klasse TreeSelectionEvent

→ Package `javax.swing.event`

→ Methoden

- `Object getSource()` Liefert das Objekt, das den Event ausgelöst hat, hier also den JTree.
- `TreePath getPath()` Liefert den ersten selektierten Pfad, der in der Pfadliste gespeichert ist. Im Modus `SINGLE_TREE_SELECTION` ist immer ein selektierter Pfad vorhanden.
- `TreePath[] getPaths()` Liefert alle selektierten Pfade (`CONTIGUOUS_TREE_SELECTION` und `DISCONTIGUOUS_TREE_SELECTION`)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

345

Klasse JTree - TreeCellEditor

× Die Datenwerte zu den Knoten können geändert werden

→ Dazu benötigt man ein Objekt, welches das Interface `TreeCellEditor` implementiert

→ Java stellt dazu eine Klasse `DefaultCellEditor` zu Verfügung

- Als `Editor` verwendet diese Klasse eine `JTextField`, eine `JComboBox` oder eine `JCheckBox`

- Konstruktoren

```
DefaultCellEditor(JTextField)  
DefaultCellEditor(JComboBox)  
DefaultCellEditor(JCheckBox)
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

346

Klasse JTree - TreeCellEditor

× Vorgehensweise

→ JTree muss zum editieren freigegeben werden
`<JTree>.setEditable(true)`

→ Klasse `DefaultCellEditor` instanziiieren

→ Diese Instanz wird dann beim JTree mit der Methode

`<JTree>.setCellEditor(Editorinstanz)`

angemeldet

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

347

Klasse JTree - TreeCellEditor

× Selektiert man einen Knoten wird der Editor aktiviert und man kann einen neuen Wert eingeben.

→ der geänderte Wert wird von JTree an das TreeModel weitergegeben. Der Knoten im Model wird dadurch ebenfalls geändert.

→ Die Änderung des Knoten wird über einen `Event` an alle registrierten `ModelListener` gemeldet

→ Beispiel

■ [..\..\..\jproject\Mehrwegbaum\Mehrwegbaum.jpx](#)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

348

DinonaRap um Soundeffekte erweitern

- ✖ Einbinden von Geräuschen
 - Bei Auswahl einer Taste
 - Beim Schießen
 - Bei Spielstatus gewonnen und verloren
- ✖ Realisierung mit Java Sound API

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

349

Java Sound API

- ✖ LowLevel.Api → `javax.sound.*`
- ✖ Ermöglicht Wiedergabe, Verarbeitung und Aufnahme von Audioformaten
- ✖ Unterscheidet
 - **Sampled Sound** (Digitalisierte Musik, Sprache und Töne)
 - `javax.sound.sampled.*`
 - **Midi** (Musik) → `javax.sound.midi.*`
- ✖ Unterstützt nicht alle audio-Formate
 - Unterstützte Soundformate:
 - AIFF, AU, WAV (nur unkomprimiert)
 - mp3 wird nicht unterstützt

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

350

Paket javax.sound.sampled

- ✖ Was kann man machen ?
 - ➔ Öffnen von input and output devices wie LineIn und LineOut der Soundkarte
 - ➔ Verwalten von (un-)buffered audio streams z.B. Audio Dateien
 - ➔ Mixen von Audiosignalen
- ✖ Die Klasse `AudioSystem` enthält Hilfsmittel für den Umgang mit aufgezeichnetem Sound
- ✖ Klasse `AudioFormat` beschreibt das Format von aufgezeichnetem Sound:
 - ➔ Kodierungsverfahren
 - ➔ Anzahl der Kanäle
 - ➔ Sampling Rate
 - ➔ Auflösung der einzelnen Samples
- ✖ Klasse `SourceDataLine` versorgt einen Mixer mit Eingabedaten (Datensenke)
- ✖ Klasse `TargetDataLine` empfängt Daten von einem Mixer (Datenquelle)

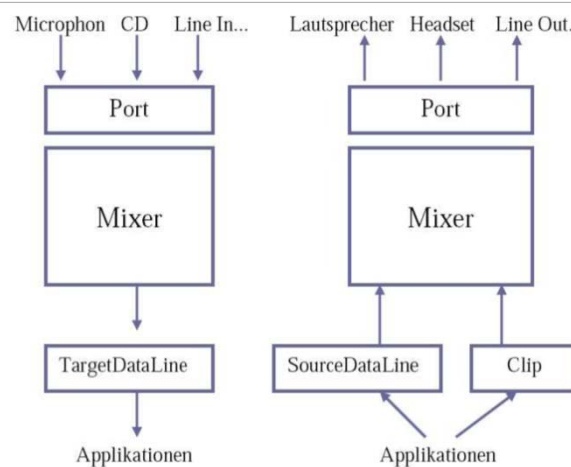
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

351

DataLines und Mixer

Quelle: Prof. Dr. Hermann de Meer, Dipl. Inf. Ivan Dedinski Sommercamp 2005, Voice over IP



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

352

Datenformate

- × Klasse **AudiFormat** repräsentiert das Datenformat der Audiodaten mit folgenden Eigenschaften
 - Encoding Technik, pulse oder modulation (PCM)
 - Anzahl der Kanäle (1 für mono, 2 für stereo)
 - Sample-Rate (Anzahl der Samples per Sekunde, per Kanal)
 - Anzahl der Bits per Sample
 - Framerate
 - Framegröße in Bytes
 - Byte order (big-endian or little-endian)
- × Klasse **AudiFileFormat** repräsentiert das Dateiformat, in dem die Audiodaten gespeichert sind. Das Datenformat hat folgende Eigenschaften:
 - Filetyp (WAVE, AIFF, etc.)
 - Filelänge in Bytes
 - Anzahl der Frames in der Datei
 - AudioFormat-Objekt, beschreibt das Format der Audiodaten

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

353

Sound abspielen

- × 2 Möglichkeiten
 - Mit einem **Clip-Objekt**
 - Die Daten werden komplett in den Speicher geladen und dann abgespielt (preload)
 - Geeignet für kleine Audio-Dateien und Audio, das öfter wieder verwendet werden soll
 - Mit einem **SourceDataLine-Objekt**
 - Die Daten werden online geladen und nur teilweise in den Speicher geladen. Während dem Abspielen werden kontinuierlich Audio-Daten nachgeladen
 - Geeignet für Streaming-Daten (Realzeitübertragung) und sehr große Audio-Dateien

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

354

Sounddateien laden und als Clip abspielen

- ✖ 1. Schritt: File-Objekt für Audiodatei erzeugen
 - ➔ `File f=new File(„klick.wav“);`
- ✖ 2. Schritt: AudioInputStream erzeugen
 - ➔ `AudioInputStream ais =`
`AudioSystem.getAudioInputStream(f);`
- ✖ 3. Schritt: Clip-Objekt erzeugen
 - ➔ `Clip audioclip = AudioSystem.getClip();`
- ✖ 4. Schritt: Clip laden mit AudioInputStream
 - ➔ `audioclip.open(ais);`
- ✖ 5. Schritt: Clip starten
 - ➔ `audioclip.start()`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

355

Links zu mehr Informationen

- ✖ Java Sound Website:
 - ➔ <http://java.sun.com/products/java-media/sound/>
- ✖ Java Sound Tutorials:
 - ➔ <http://java.sun.com/docs/books/tutorial/sound/>
- ✖ Java Sound Resources:
 - ➔ <http://www.jsresources.org/>
- ✖ Java Sound Programmer Guide:
 - ➔ <http://java.sun.com/j2se/1.4.2/docs/guide/sound/>
- ✖ Java Sound FAQ:
 - ➔ <http://jsresources.org/faq.html>
- ✖ Java Sound API:
 - ➔ <http://java.sun.com/j2se/1.4.2/docs/api/javax/sound/sampled/>
 - ➔ <http://java.sun.com/j2se/1.4.2/docs/api/javax/sound/midi/>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

356

Java Sound Api

✖ [..\audio\Java2Sound.pdf](#)

Java Web Start

- ✖ Wozu wird Web Start verwendet ?
 - ➔ Starten einer Java-Anwendung, die auf einem Web-Server abgelegt ist, übers Internet durch Aufruf **in einem beliebigen Web-Browser**
 - ➔ Verwendung von swing-Komponenten in Java-Applets
 - ➔ Automatische Installation der notwendigen Java-Version übers Internet
 - ➔ Die Anwendung läuft voreingestellt in einer **sicheren Umgebung** (**secure sandbox**) ab, dh. Zugriffe auf das lokale System und Netzwerk sind nur eingeschränkt möglich
 - ➔ Einmal geladene Anwendungen werden lokal gespeichert und müssen bei weiteren Aufrufen nicht wieder geladen werden

Java Web Start

× Bedingungen ?

→ Client:

- Java Runtime Umgebung Version 1.2.2 oder höher
- Betriebssysteme Windows 95/98/NT/2000/XP oder ME, Red Hat Linux oder Solaris
- Die Java-Anwendung wird über die Web Start Software vom Webserver geladen und gestartet. Daher muss Web Start installiert sein - ab JDK 1.4 schon enthalten

→ Server

- muss konfiguriert sein für den **mime-type** `application/x-java-jnlp-file`

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

359

Java Web Start

× Anforderungen an das Java-Programm

- Zum Anwendungsprogramm muss eine **jnlp-Datei** existieren. Sie beschreibt der Web Start Software die Anwendung.
- Die Anwendung muss als Bytecode in ein oder mehreren Archiven (**jar-Dateien**) auf einem Web Server zu Verfügung stehen
- Alle zusätzlichen Ressourcen wie Dateien und Bilder müssen sich in den jar-files befinden und mit der Methode `getResource()` aus dem Archiv geladen werden
- Alle jar-Files müssen auf dem gleichen Server liegen
- Das Programm darf nicht auf die lokale Platte zugreifen und hat eingeschränkten Zugriff auf System-Eigenschaften
- Nur **signierte jar-Files** haben uneingeschränkten Zugriff auf das lokale System. Dazu müssen jedoch alle Elemente in allen jar-Files signiert sein.
- Netzwerkzugriffe sind nur auf den Server erlaubt, auf dem die jar-Files liegen
- Es dürfen **keine "native Libraries"** verwendet werden

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

360

Java Web Start

× Was ist zu tun ?

- Es muss ein **jar-File** erzeugt werden, in dem alle Java-Klassen und alle vom Programm benötigten benötigten Ressource-Dateien gespeichert werden
- Ressourcen müssen im Java-Programm aus dem jar-File geladen werden
- Es muss eine **jnlp-Datei** angelegt und auf dem Web Server gespeichert werden. In dieser xml-Datei werden die Eigenschaften des Java-Programms beschrieben, damit Web Start die Anwendung laden und starten kann
- Es muss eine **html-Seite** geben, in der ein Link auf die jnlp-Datei eingetragen ist. Über diesen Link wird Web Start und über dieses dann das Java-Programm gestartet.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

361

Java Web Start

× Weiterführende Informationen sind zu finden unter:

<http://java.sun.com/products/javawebstart/developers.html>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

362

Das Java- Archiv

- ✖ eine **jar-Datei** ist ein **Archiv**, in dem mehrere class-Dateien zusammen gefasst werden. Die Anwendung kann damit in einer Datei zu Verfügung gestellt und einfach durch Doppelklick gestartet werden
- ✖ Das Archiv kann ungepackt oder mit Hilfe vom ZIP-Algorithmus **gepackt** werden. Gepackt verringert sich die Ladezeit übers Netz.
- ✖ Eine jar-Datei enthält alle Klassen, Bild- und Audiodateien sowie sonstige Datendateien der Anwendung

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

363

Das Java- Archiv

- ✖ Ein Archiv enthält normalerweise eine **Manifest-Datei**, die den Inhalt des Archivs beschreibt. Diese Manifestdatei muss in einem Verzeichnis META-INF liegen und den Namen MANIFEST.MF haben.
 - ➔ Für eine Standalone-Anwendung steht in der Manifestdatei die Hauptklasse (**mainclass**), in der die Methode main() zu finden ist.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

364

Das Java- Archiv

→ Erstellt wird das Archiv mit dem **jar-Tool** des JDK

■ Syntax: **jar** [options] [manifest] jar-file input-file [input-files]

Gibt man als Dateiname ein Verzeichnis an, so wird das Verzeichnis rekursiv mit allen Dateien und Unterverzeichnissen ausgewertet

■ Mögliche Optionen:

- c erzeuge ein neues Archiv
- t Inhaltsverzeichnis des Archivs ausgeben
- x extrahiere angegebene Datei oder alle Dateien
- f Name der jar-Datei wird angegeben
- m Füge Manifest-Informationen aus dem angegebenen Manifest- Datei ein

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

365

Das Java- Archiv

→ Weitere Informationen zum jar-Archiv:
<http://java.sun.com/j2se/1.3/docs/tooldocs/win32/jar.html>

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

366

Das Java- Archiv

- Beispielprogramm "lottoGrafik"

Archiv lottoGrafik.jar enthält die Dateien:

```
lottografik\liste.class  
lottografik\lottoGrafik.class  
lottografik\ziehungszahlen.class  
META-INF/MANIFEST.MF
```

Inhalt der Manifest-Datei:

```
Manifest-Version: 1.0  
Main-Class: lottografik.lottoGrafik
```

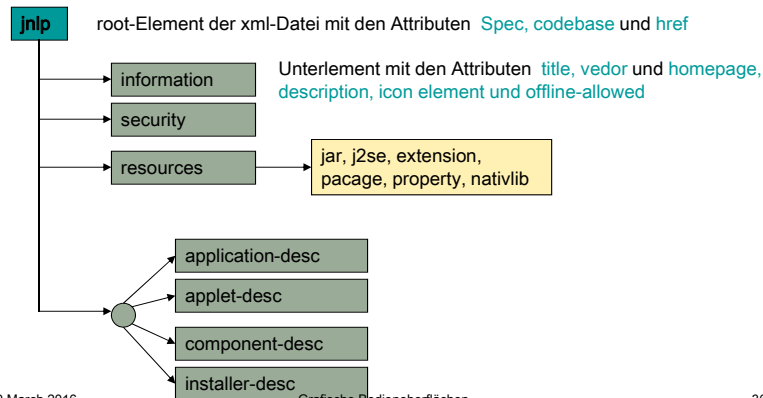
Das Java- Archiv

- In Eclipse erstellt man eine Archiv über den Menüpunkt:

Datei → Export → java → jar-File

Web Start – jnlp-Datei

✖ xml-Datei mit folgender Struktur



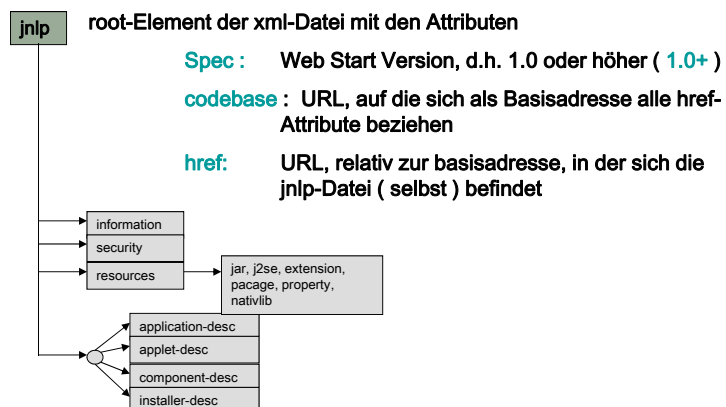
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

369

Web Start – jnlp-Datei

jnlp-Element

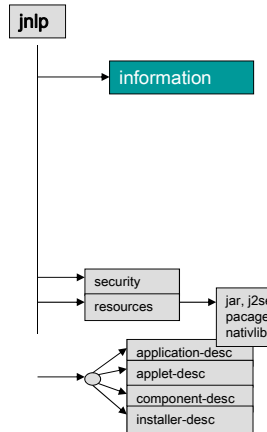


10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

370

Web Start – jnlp-Datei - information- Element



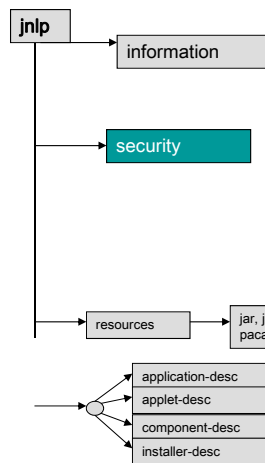
10 March 2016

Unterelemente

- title:** Name der Anwendung
- vedor :** Vertreiber der Anwendung
- homepage :** URL für ein html-Doument, das weitere Informationen zu der Anwendung enthält
- description :** Eine Kurzbeschreibung zu der Anwendung. Dies kann sein ein tooltip, eine mehrzeilige Beschreibung oder eine einzeilige Beschreibung
- icon element:** URL zu einem leinen Bild, das in Web Start angezeigt wird während das Programm geladen wird
- offline-allowed:**
- Ist diese Element nicht vorhanden muss der Klient online sein. Es wird dann bei jedem Start auf dem Server geprüft, ob es neuere Versionen gibt. Wenn ja werden dies automatisch geladen.
- Ist dieses Element vorhanden, kann die Anwendung auch gestartet werden wenn der Klient nicht online ist. Web Start sucht auch in diesem Fall auf dem Server nach einer jüngeren Version. Reagiert der Server nicht in einer bestimmten Zeit geht Web Start davon aus, das die Anwendung offline ist und nimmt die lokal gespeicherte Version.

Grafische Bedienoberflächen
Prof. S. Keller

Web Start – jnlp-Datei – security-Element



10 March 2016

Dieses Element hat entweder kein weiteres Element oder das Unter-Element **all-permission**

Ohne ein Element läuft die Anwendung in einer **sicheren** Umgebung ab

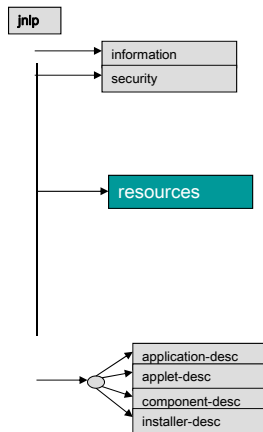
Ist das Element all-permission angegeben hat die Anwendung alle Rechte und kann damit auch auf die lokalen Dateien zugreifen. Dann müssen jedoch alle jar-Files und alle Elemente in den jar-Files **signiert** sein.

Ist dies der Fall muss der Benutzer das Zertifiat akzeptieren, damit Web Start die Anwendung startet

Grafische Bedienoberflächen
Prof. S. Keller

372

Web Start – jnlp-Datei - ressourcen



Mit diesem Element werden alle benötigten Ressourcen angegeben
Es sind folgende Unterelemente möglich

jar: Gibt die relative URL eines benötigten jar-File an. Diese jar-File wird mit einer Instanz der Klasse ClassLoader in die VM geladen

j2se: Gibt die notwendige Version der Java 2 Runtime Environment (JRE) eventuell mit notwendigen Parametern

`<j2se version="1.3" initial-heap-size="64M"/>`

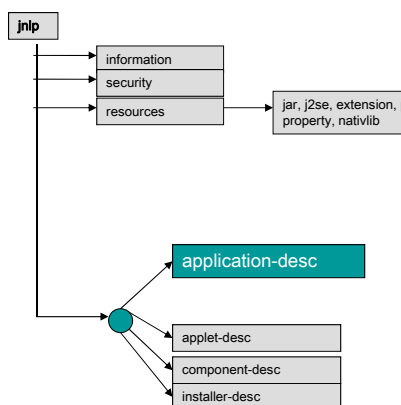
property: Gibt an welche System-Eigenschaften von dem Programm benötigt werden (mit der Methode System.getProperty bzw. System.getProperties)

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

373

Web Start – jnlp-Datei - Application



Mit dem Element **application-desc** wird Web Start angegeben, dass es sich um ein eigenständige Anwendung handelt.

Über das Attribut main-class wird angegeben, in welcher Klasse des jar-Files sich die Methode main() befindet.

Mit dem Element **applet-desc** wird Web Start angegeben, dass es sich um ein Applet handelt.

Über weitere Parameter wird das Applet näher beschrieben analog dem html-tags zu einem Applet.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

374

Web Start – jnlp-Datei Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File fuer Dezimal-Dualzahl-Rechner -->
<jnlp
  spec="1.0+"
  codebase="http://erde.fbe.fh-weingarten.de/keller"
  href="grabo/swingRechner.jnlp">
  <information>
    <title>Dezimal-Dualzahl-Rechner</title>
    <vendor>FH Ravensburg-Weingarten, Prof. Dr. S. Keller.</vendor>
    <homepage href="http://erde.fbe.fh-weingarten.de/keller/grabo.html"/>
    <description>Dezimal-Dualzahl-Rechner</description>
    <description kind="short"> Eine swing Anwendung </description>
    <icon href="grabo/fh.gif"/>
    <icon kind="splash" href="grabo/fh.gif"/>
    <offline-allowed/>
  </information>
  <security>
  </security>
  <resources>
    <j2se version="1.3"/>
    <jar href="grabo/swingRechner.jar"/>
  </resources>
  <application-desc main-class="swingRechner"/>
</jnlp>
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

375

Java Web Start – Laden von Bildern aus dem Archiv

× Beispiel - Laden eines Bildes aus einem jar-File

- Im Archiv wurde die Bild-Datei fh.gif eingefügt
- Um eine Ressource aus dem Archiv zu laden wird die Instanz der Klasse ClassLoaders benötigt, die die Klassen aus dem jar-File der VM übergibt

```
ClassLoader loader=this.getClass().getClassLoader();
```

- Um die Bild-Datei aus dem jar-File zu laden, muss die Methode getResource() des ClassLoaders verwendet werden d.h. anstatt der Anweisung

```
bild=getToolkit().getImage(dateiname);
```

muss die Anweisung so lauten:

```
bild=getToolkit().getImage(loader.getResource(dateiname));
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

376

Java Media Framework (JMF)

- × JMF Working Group
 - Sun, Silicon Graphics, Intel
- × Framework zur Integration kontinuierlicher Medien in Javaprogramme
- × Ermöglicht die
 - Präsentation
 - Verarbeitung, Transformation
 - und Synchronisation kontinuierlicher Medien

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

377

Java Media Framework (JMF)

- × Was ist ein Framework ?
- × Was sind kontinuierliche Medien ?

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

378

Was ist ein Framework ?

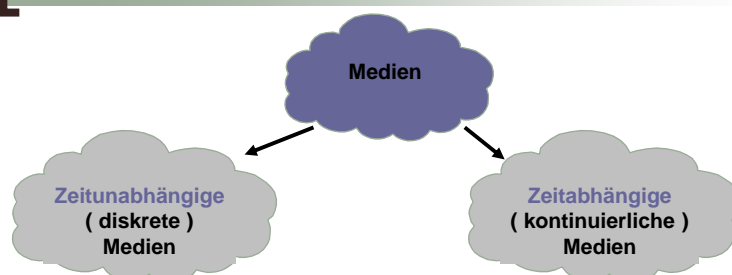
- ✖ Gibt eine Softwarearchitektur für einen bestimmten Anwendungsbereich vor
- ✖ Objektorientierte Applikationsbaukästen
 - Klassenbibliotheken
 - Schnittstellendefinitionen
 - Werkzeuge

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

379

Kontinuierliche Medien



- ✖ Die Information wird durch zeitkonstante Werte vermittelt
 - Beispiele: Text, Grafik, Bild
- ✖ Die Verarbeitung ist zeitunkritisch
- ✖ Die Information wird durch eine Werteänderung mit der Zeit, also einem zeitlichen Verlauf vermittelt.
 - Beispiel: Filme, Musik
- ✖ Die Verarbeitung ist zeitkritisch

* aus Steinmetz: Multimedia-Technologie: Einführungen und Grundlagen
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

380

- ✖ Wiedergabe von kontinuierlichen Medien
 - ➔ Realisierung von Media-Player
- ✖ Übertragung von Echtzeitdatenströmen (Streaming Media) über das Netzwerk
 - ➔ Broadcasting, Internettelefonie
 - ➔ Unterstützt RTP (Realtime Transport Protocol)
- ✖ Transformation und Verarbeitung von Medienformaten
- ✖ Ermöglicht eine Synchronisation von Medien

- ✖ Unterstützt viele verschiedene Übertragungsmechanismen und Protokolle
- ✖ Unterstützt viele verschiedene Medientypen
- ✖ Ereignismodell für eine asynchrone Kommunikation zwischen Media Player und Java-Anwendungen

Java Media Framework (JMF)

× Unterstützte Formate

- Video
 - MPEG-1, MPEG-2, MPEG-4, AVI, Quicktime, ...
- Audio
 - WAV, AU, MP3,...
 - MIDI
- Animationen
 - Flash
- Bilder
 - JPEG, ...

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

383

Java Media Framework

× Prozessmodell



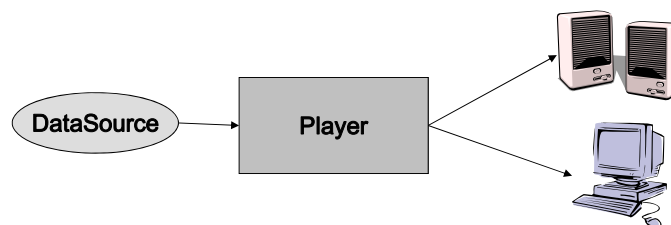
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

384

- ✖ Objekte, die zum Transfer der Mediendaten verwendet werden
- ✖ kapselt Protokoll und Position der Medien
- ✖ Arten
 - **pull data source**
 - der Client fängt mit dem Datentransfer an (http, Datei)
 - onDemand, Positionierung möglich
 - **push data source**
 - der Server sendet und kontrolliert Datentransfer (Steaming media - RTP)
 - Broadcasting, Internettelefonie, Liveübertragung, keine Positionierung möglich

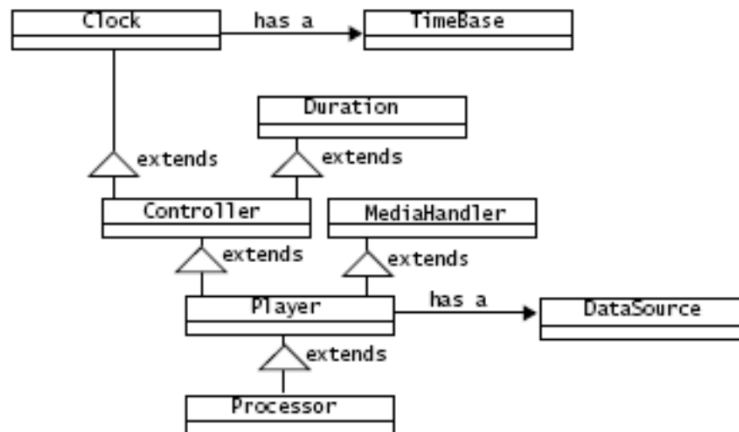
- ✖ verarbeitet einen Strom von Mediendaten



- ✖ rendert Medium in definierter Zeit
- ✖ Kontrolle erfolgt über Clock, Controller

Klassen

Player



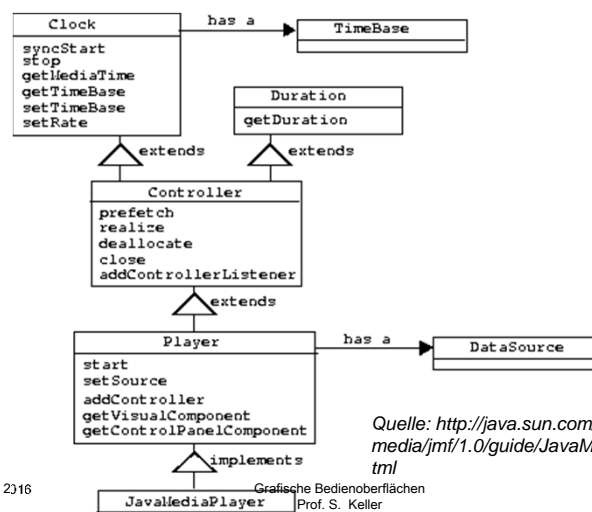
10 March 2016

Quelle: Java® Media Framework API Guide, November 19, 1999, JMF 2.0
FCS, Sun Microsystems, Inc. Prof. S. Keller

387

Klassen

Player



10 March 2016

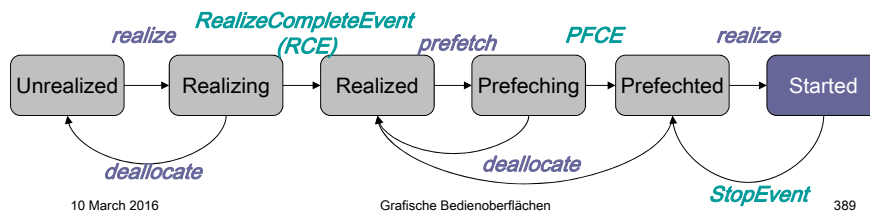
Quelle: <http://java.sun.com/products/java-media/jmf/1.0/guide/JavaMediaFrame.fm1.html>

Grafische Bedienoberflächen
Prof. S. Keller

388

Java Media Framework Player Zustände

- × Zwei Hauptzustände
→ Stopped / Started
- × Zustand Stopped unterteilt sich in weitere Zwischenzustände
- × Zustandsübergänge durch Methodenaufrufe
- × Zustandswechsel werden durch **TransitionEvents** signalisiert



Java Media Framework Player Zustände

- × **Unrealized**
→ Player wurde instanziiert aber kennt seine DataSource noch nicht
- × **Realizing**
→ Player sucht alle seine benötigten Ressourcen und ermittelt den Typ des abzuspielenden Mediums
- × **Realized**
→ Player kennt alle benötigten Ressourcen und kennt den Typ des abzuspielenden Mediums
→ Erzeugt **visuelle Objekte und Komponenten**, die angezeigt werden können
- × **Prefetching**
→ Mediendaten werden in den Speicher geladen
- × **Prefetched**
→ Mediendaten sind geladen und können wiedergegeben werden
- × **Started**
→ Daten werden abgespielt

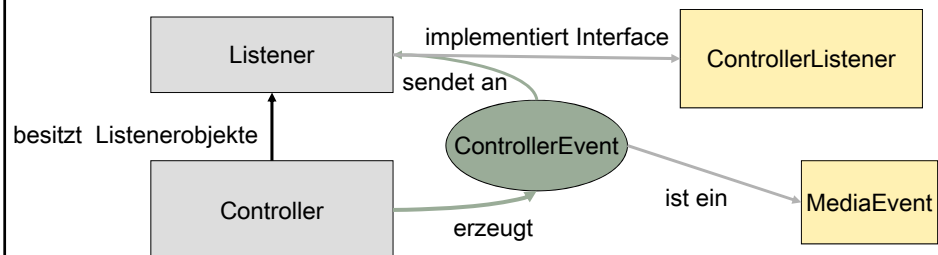
10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

390

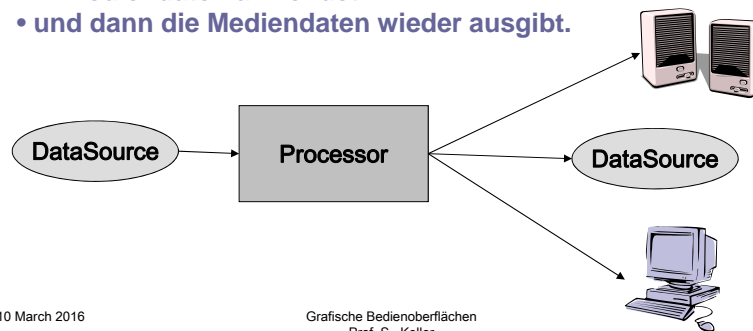
- ✖ Verwaltet alle Systemressourcen
- ✖ Überwacht den zeitlichen Ablauf der Medien und erzeugt entsprechende Ereignisse (Events)

- ✖ Analog der Eventverarbeitung in Java
- ✖ Controller erzeugen ControllerEvents
- ✖ Implementierung des Interfaces ControllerListener um auf Events zu reagieren



Ein Processor ist ein Player

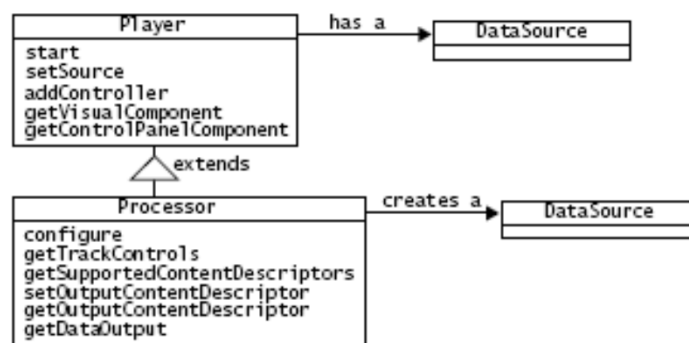
- der Mediendaten einliest,
- benutzerdefinierte Verarbeitungsschritte auf die Mediendaten anwendet
- und dann die Mediendaten wieder ausgibt.



10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

393



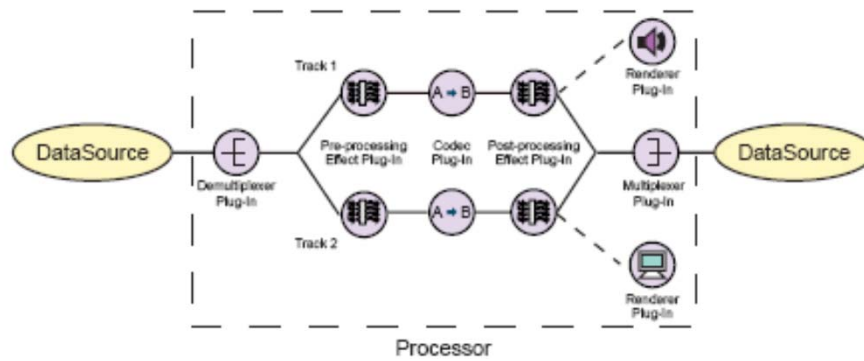
Quelle: Java® Media Framework API Guide, November 19, 1999, JMF 2.0
FCS, Sun Microsystems, Inc.

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

394

Java Media Framework Processor



Quelle: Java[®] Media Framework API Guide, November 19, 1999, JMF 2.0

10 March 2016

FCS, Sun Microsystems Inc.
Grafische Bedienoberflächen
Prof. S. Keller

395

Java Media Framework Manager

- ✖ JMF besteht zum größten Teil aus Interfaces
- ✖ Manager implementieren diese Interfaces
- ✖ Klassenobjekte, mit denen man Playerobjekte und Processoren erzeugen kann

→ Beispiel:

```
Player player = Manager.createPlayer(url);
```

10 March 2016

Grafische Bedienoberflächen
Prof. S. Keller

396