



JAVA

Oberflächen mit Swing

Grafik von <http://javaafx.com>

168



AGENDA



- Grundlagen GUI
- Swing
- JavaFX

GRAPHISCHE BENUTZEROBERFLÄCHEN (I)

- AWT = Abstract Window Toolkit
 - Verwendet die jeweiligen GUI-Komponenten des OS
 - Bietet nur eine Grundmenge von Elementen aller OS
- Swing
 - Swing werden direkt von Java gerendert und sind OS frei
 - Mehr Elemente und bequemere API
 - Oft kein natives OS Verhalten und etwas langsamer

170

- 2. - Wegen Swing sind Java Desktop Applikationen so in Verruf geraten.
 - Man merkt den Applikationen eben an, dass es Java ist.
- 0. - Weitere Oberflächentechnologien von Java:
 - SWT -> Standard Window Toolkit
 - Von eclipse (entwickelt und verwendet)
 - Verwendet native UI Komponenten, reduziert sich jedoch auf den kleinsten gemeinsamen Nenner aller.
 - Rest wird in pure Java implementiert.

GRAPHISCHE BENUTZEROBERFLÄCHEN (2)



- JavaFX
 - Ab Java 7 Teil vom JDK
 - Aktuelle Version 8
 - Portabel
 - Direkt von Java gerendert

171

1. - Kann aber auch separat heruntergeladen werden
2. - JavaFX 1.x kann nicht mit 2.x verglichen werden.
 - 2007 hat Sun JavaFX offiziell angekündigt, war aber sehr experimentell.
 - 2008 kam JavaFX 1.0 als Entwicklerkit für Mac und Windows.
 - 2009 kam Version 1.2 mit mehr UI Komponenten und Linux support, aber keiner Abwärtskompatibilität.
 - Bisher war es nicht so wirklich gut benutzbar und nie abwärtskompatibel nach einem Release.
 - 2011 kam Version 2.0 und es wurde bisheriger Scriptansatz verworfen und aktuelle Strategie implementiert.
 - 2.0 unterstützt Hardware-Rendering
3. - JavaFX kann auf dem Desktop ausgeführt werden, allerdings auch als WebAnwendung auf den Client herunter geladen werden.
 - Weiterhin werden JavaFX im Browser laufen, auf SetTop Geräten, Mobilfunkgeräten, ...
 - Eher der webbasierte Weg
 - UI Komponenten können per CSS gestyled werden.
4. - Ähnlich zu Swing.
 - Aber können mit unterschiedlichem Look and Feel für unterschiedliche Betriebssysteme angezeigt werden.

SWING

- Alle Swing UI Komponenten sind `java.awt.Container` und können andere Komponenten beinhalten
- Ermöglicht ein einheitliches Konzept zum Rendering
- Scope dieses Abschnittes: Grundkonzept kennen

172

1. - Baut auf AWT auf, verwendet einige AWT Klassen
- Aber: Komplette in Java entwickelt, verwendet keine Betriebssystemressourcen
- Pluggable Look-and-Feel
- Plattformunabhängig, da Rendering über Java Komponenten
3. - <http://docs.oracle.com/javase/tutorial/uiswing/components/index.html>
- <http://docs.oracle.com/javase/tutorial/ui/features/components.html>

SWING JCOMPONENT

- JComponent ist die Basisklasse aller Swing UI Komponenten
- Erbt von `java.awt.Container` -> Alle Swing Elemente Container
- JComponent unterstützt viele Features
 - ToolTips, Border, Pluggable Look & Feel, Drag & Drop

173

2.

- Ein Button kann z.b. ein Icon und ein Text beinhalten

HALLO SWING

```
package ba.java.swing;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class SwingApp {
    ... public SwingApp() {
        ..... JFrame frame = new JFrame("Erste Swing Applikation");
        ..... frame.getContentPane().add(new JLabel("Hallo!"));
        ..... frame.setSize(400, 200);
        ..... // frame.pack(); // für automatische Größenberechnung
        ..... frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ..... frame.setVisible(true);
    }
    ...
    ... public static void main(String[] args) {
        ..... new SwingApp();
    }
}
```



SWING KOMPONENTEN

- JFrame
- JComponent
- JPanel
- JScrollPane
- JButton
- JLabel
- JTabbedPane
- JCheckBox
- JComboBox
- JTextField, JTextArea
- JPasswordField
- JSlider, JMenuBar, ...

175

Vergleiche: <http://docs.oracle.com/javase/tutorial/ui/features/components.html>

SWING

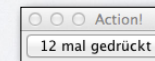
EVENTS & LISTENER

- Package java.awt.event bzw. java.swing.event
- java.awt.ActionListener ist die Mutter aller EventListener
- Meist anonyme Klasse, die Interface implementiert

```
public class DrueckMich {
    ... private int count = 0;
    ... private final String GEDRUECKED = " mal gedrückt";

    ... public DrueckMich() {
        ... JFrame derFrame = new JFrame("Action!");
        ... final JButton derButton = new JButton(count + GEDRUECKED);
        ... derButton.addActionListener(new ActionListener() {
            ... @Override
            ... public void actionPerformed(ActionEvent e) {
            ...     count++;
            ...     derButton.setText(count + GEDRUECKED);
            ... }
        });
        ... derFrame.getContentPane().add(derButton);
        ... derFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ... derFrame.pack();
        ... derFrame.setVisible(true);
    }

    ... public static void main(String[] argv) {
    ...     new DrueckMich();
    ... }
}
```



176

2. - Der ActionListener reagiert auf alle Arten von Events.
- Meistens macht es Sinn nur auf bestimmte Events zu reagieren, dann nimmt man einen spezielleren Listener, z.b. MouseListener.
0. - Frage: Warum ist „derButton“ als final deklariert?

SWING

LISTENER UND ADAPTER

- Für jeden Typ von Listener existiert ein Interface, das den Listener definiert. Namensschema: `<Typ>Listener`
- Dazu gibt es jeweils eine passende Klasse, die die Event Informationen hält. Namensschema: `<Typ>Event`
- Auf Komponenten existiert Methode `add<Typ>Listener(..)`
- Meist existiert eine Adapterklasse, Schema: `<Typ>Adapter`

177

- 4.
- Falls mehr als eine Methode im Listener Interface definiert ist, dann existiert meistens zusätzlich eine abstrakte Adapterklasse, die das Interface mit leeren Methoden implementiert.
 - Warum könnte das sinnvoll sein?
 - Beispiel: `MouseListener` definiert 5 Methoden, aber vielleicht interessiert mich nur `mouseClicked`?
 - Welchen Nachteil hat die Verwendung von Adapterklassen?
 - Man verbaut sich den Weg von einer anderen Klasse zu erben.
 - Die Klassen die auf etwas reagieren, sind meistens mehr als „nur“ Listener
 - > anonyme Überschreibung von Adapterklassen ist daher ratsam.

SWING

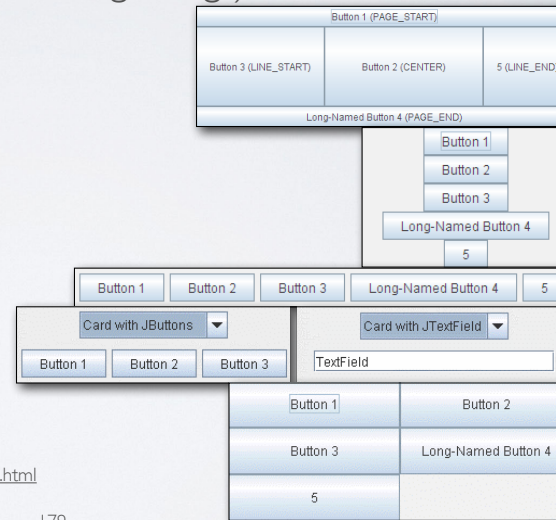
PROMINENTE EVENTLISTENER

- WindowListener
- ActionListener
- KeyListener
- MouseListener
- MouseMotionListener

SWING LAYOUTMANAGER

- Container#setLayout(LayoutManager mgr)
- BorderLayout
- BoxLayout
- FlowLayout
- CardLayout
- GridLayout

<http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

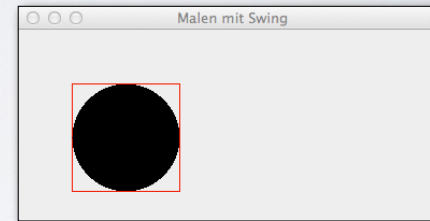


179

0. - Vergleiche <http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
1. - Ein BorderLayout teilt den Container in bis zu 5 Teile (Top, Bottom, Left, Right, Center bzw. N, E, S, W, Center).
- Aller Platz, der übrig bleibt geht in den Center Bereich.
2. - BoxLayouts setzen alle Komponenten in eine Reihe oder eine Spalte.
- Elemente können innerhalb der Reihe/Spalte ausgerichtet werden (left, right, center).
3. - FlowLayout stellt alle Elemente in einer Reihe dar.
- Ein Zeilenumbruch kommt, wenn der Container zu klein wird in der horizontalen.
4. - CardLayout stellt einen Manager dar, der verschiedene Elemente zu verschiedenen Zeitpunkten darstellt.
- Dabei ist immer nur eine Variante von Components zu einer Zeit sichtbar.
- Funktionalität wie ein JTabbedPane.
5. - Elemente werden innerhalb einer definierten Anzahl Zeilen und Spalten angeordnet.
- Elemente haben alle die gleiche Größe.

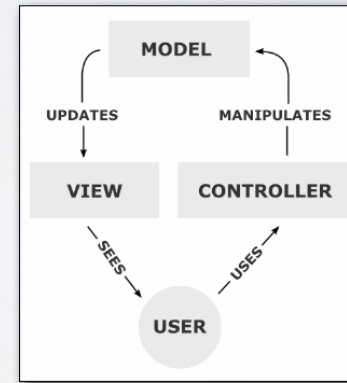
SWING ZEICHNEN

```
public PaintBeispiel() {  
    ... JFrame frame = new JFrame("Malen mit Swing");  
    ... frame.setSize(400, 200);  
    ... JPanel zeichenFläche = new JPanel() {  
  
        // jede Swing JComponent hat eine methode paint(),  
        // welche zum Zeichnen überschrieben werden kann.  
        @Override  
        public void paint(Graphics g) {  
            super.paintComponents(g);  
            g.setColor(Color.RED);  
            g.drawRect(50, 50, 100, 100);  
            g.setColor(Color.BLACK);  
            g.fillOval(50, 50, 100, 100);  
        }  
    };  
    frame.getContentPane().add(zeichenFläche);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
}
```



JAVAFX

- Ähnlich zu Swing, aber komfortabler
- Superklasse Node bzw. Control
- MVC
- Sourcen (sind bei Java 8 enthalten)

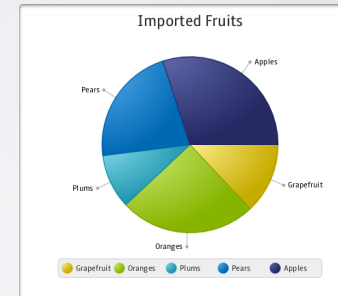


Grafik von <http://en.wikipedia.org/wiki/File:MVC-Process.png>

181

1. - Konzept der Layout Manager ist vorhanden, allerdings etwas moderner.
- EventListener sind analog zu Swing, allerdings auch hier etwas moderner.
- Eingabeelemente sind ähnlich, aber auch mit modernerer API.
2. - Ähnlich zu Swing, allerdings anderer Name.
3. - Stärkeres MVC Pattern als bei Swing.
- Es gibt Models, Views und Controller.
- Model = Datenhaltung.
- Views = Präsentationsschicht.
- Controller = Enthalten eigentliche Logik.
- Wird gleich im Beispiel gezeigt.

JAVAFX GUI ELEMENTE



182 Grafik von http://docs.oracle.com/javafx/2/ui_controls/overview.htm

Dokumentation von oracle ist sehr gut!

Siehe:

http://docs.oracle.com/javafx/2/ui_controls/overview.htm

http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

JAVAFX SCENEBuilder

- Grafischer Editor
- .fxml und @FXML
- <http://www.oracle.com/technetwork/java/javafx/tools/index.html>

183

Gezeigt während der Live-Dome:

Anlegen neues IntelliJ JavaFX Projekt

Aufbau GUI

- Library
- Hierarchy
- Scene
- Layout
- fx:id Attribut
- Code Tab würde ich erstmal nicht benutzen!

Gui Elemente

- Containers
- Layouts
- Wrap in Layout
- Controls

Größen

- Computed_Size

-- Typically, the computed size is just big enough for the control and the label to be fully visible.

- Preferred_Size

-- If you want more control over the size of controls in your UI, you can set their preferred size range directly.

FXML kann natürlich auch per Texteditor bearbeitet werden

- Wie setzen wir einen Controller
- Wie verwende ich Controls mit fx:id?

WICHTIGE DINGE ÜBER JAVAFX

- .fxml und @FXML
- javafx.fxml.Initializable
- ObservableList und FXCollections
- Inhalt eines Packages
- Dokumentation ist sehr gut

184

1. - Annotation @FXML macht autobinding von Variable zu konkretem Control in der Oberfläche mittels fx:id.
2. - Interface Initializable kann als Controller einer Oberfläche gesetzt werden.
- initialize(URL url, ResourceBundle resourceBundle) wird aufgerufen zur Initialisierung.
3. - Es ist von Haus aus möglich CSS zum Styling zu verwenden.
- Für Designer sehr angenehm, eher der webbasierte Weg.
4. - Inhalt eines Packages sollten in JavaFX sein: „Alles was zu einer View dazugehört.“
 - View Klasse
 - Controller Klasse
 - CSS Datei
 - Helfer Klassen
- Modell ist separat, da es meistens View-übergreifend ist.
5. - <http://www.oracle.com/technetwork/java/javafx/documentation/index.html>
- http://docs.oracle.com/javafx/2/ui_controls/overview.htm
- http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

ZU INSTALLIEREN

- Aktuelles Oracle JDK
- IntelliJ oder efxclipse.org -> Vollständige eclipse Distribution
- SceneBuilder

HAPPY HACKING

