Einführung in GUI-Programmierung

javax.swing

GUI – Graphical User Interface („Grafische Benutzerschnittstelle“)

- Anschaulichere und leichtere Möglichkeit zur Dateneingabe und Kontrolle von Programmen
- Stellt Textfelder, Schaltknöpfe, Menüs, usw. zur Verfügung
- Java SE stellt Bibliotheken zur einfachen Implementierung von GUIs zur Verfügung
  - Swing:
    - Früher (Java 1.2 ca. 1998): Swing als fester Bestandteil der JRE
    - Entwickelt von Sun
    - Baut auf dem älteren Abstract Window Toolkit (AWT) auf.
      - Erweiterungen: Drag & Drop, neue Panels und Layouts, weitere Komponenten
  - JavaFX:
    - Heute(Seit Java SE 7 Update 6): JavaFX als fester Bestandteil von Java SE x86
    - Mittlerweile aber wieder aus Java SE ausgegliedert
    - Schnell erstellbare neue UI-Komponenten (per CSS gestaltbar).

Zur GUI-Entwicklung in Java gibt es eigene Frameworks (Bibliotheken)

Abstract Window Toolkit

- Gehört seit Einführung (1985) zur Standard-API
- Zur Gestaltung einfacher grafischer Oberflächen
- Bietet Methoden zur Ereignisbehandlung zum Zeichnen und versch. GUI-Komponenten

Swing

- Um anspruchsvollere GUIs zu ermöglichen wurden neue Komponenten hinzugefügt
- AWT + Swing = Java Foundation Classes (JFC)
- Bis heute Teil der Java SE

Fokus heute

JavaFX

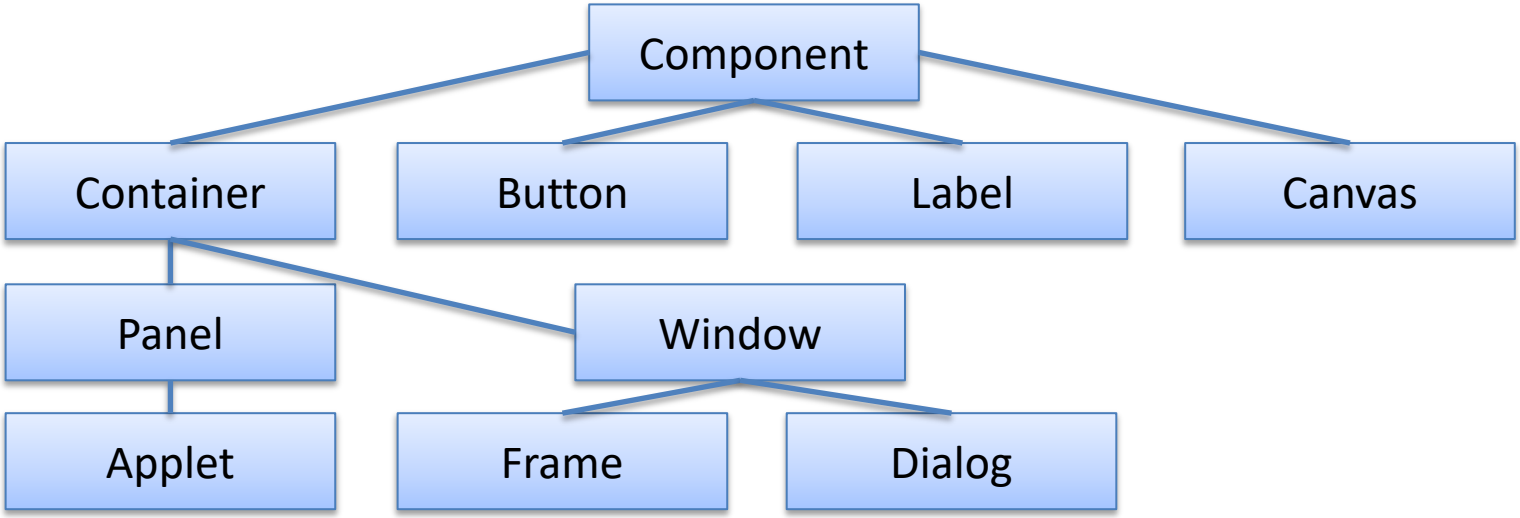
- Komplette Neuentwicklung der GUI-Ebene
- Ermöglicht kompletten Media-Stack mit Video, Audio, Animationen, 2D & 3D u.v.m.
- Lange Zeit Teil der Java SE – mittlerweile aber wieder ausgegliedert

Heute: Kurze Einführung in **Swing**

- Java Swing-API ist umfangreich und sehr flexibel
- Ist im Package `javax.swing` verankert
  - Bietet ca. 18 weitere Unterpakete:
    - `border, event, table, text, tree,` usw.

Grundlage für die grafische Entwicklung ist AWT (Abstract Window Toolkit).

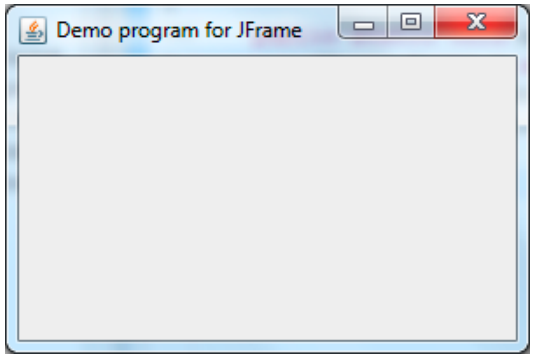
Ein Auszug aus der Klassenhierarchie (`java.awt`):



In Swing gibt es drei sogenannte Top-Level Container:

- `JFrame`: Hauptfenster bildet die Grundlage der meisten GUIs
- `JDialog`: Dialogfenster ohne Menüleiste
- `Japplet`: Webbrowser-Programme zur client-seitigen Verarbeitung

Jede Anwendung die Swing Komponenten enthält braucht mindestens einen Top-Level Container

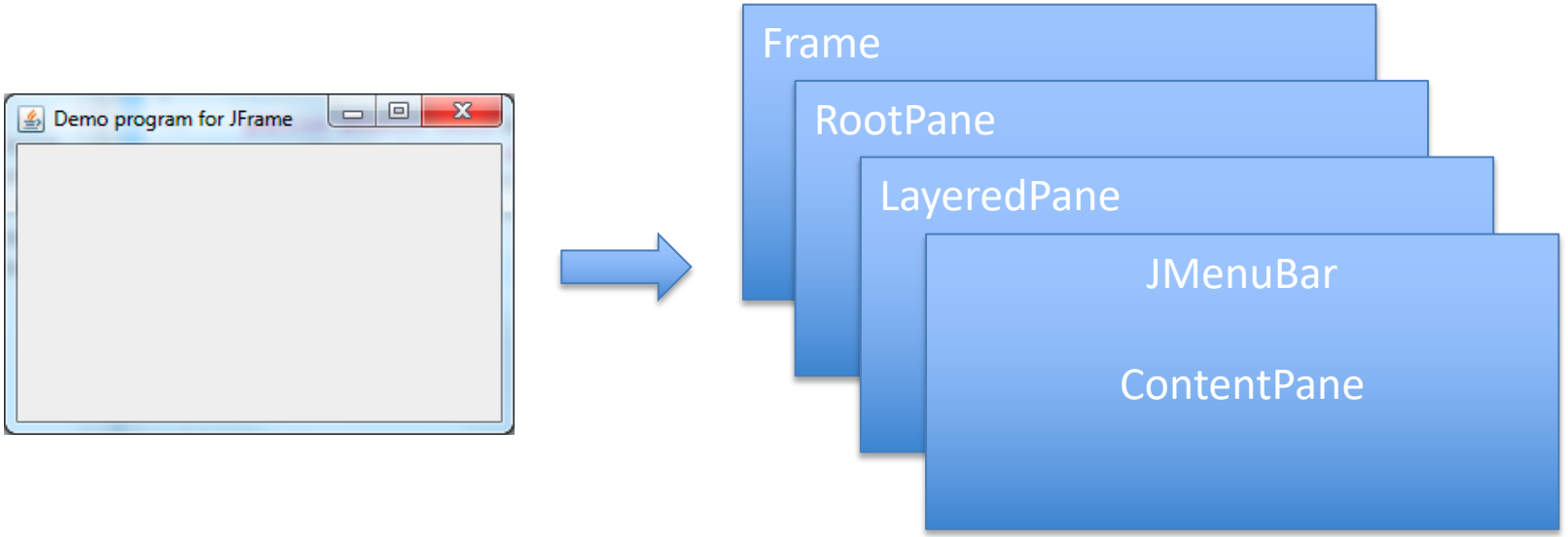


- Dieser Container ist die Wurzel einer Komponenten-Hierarchie
- Top-Level Container enthält außerdem Content-Pane, die (direkt oder indirekt) die sichtbaren Komponenten enthält

Damit eine GUI-Komponente auf dem Screen angezeigt werden kann, muss sie der Komponenten-Hierarchie hinzugefügt werden

JFrame

Allgemeiner Aufbau:



Unser `JFrame` beinhaltet das `JRootPane` als einziges Kind

- Stellt die `ContentPane` zur Verfügung
- Das `ContentPane` ist die Basis-Komponente für alle Unterkomponenten

Unserem `JFrame` können nun direkt Elemente (`Components`) hinzugefügt werden:

- `this.add(new JButton(„OK“)) // Fügt einen OK-Button hinzu`

Oder man definiert seine eigene `ContentPane` und gibt diese dem `JFrame`:

- `JPanel contentPane = new JPanel();`
- `setContentPane(contentPane);`

Fenster erstellen mit Swing

Wir wollen nun eine einfache Nutzerschnittstelle entwerfen:

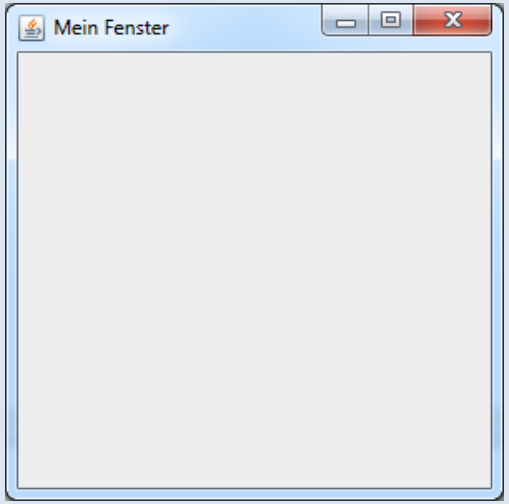
- Mit Hilfe der Klasse `JFrame` aus dem Packet `javax.swing` können wir sehr leicht ein einfaches bewegliches plattformabhängiges Fenster erstellen.
- Die Klasse bringt sehr viele Methoden und Eigenschaften mit, bspw.:
  - `setSize(width, height); // Legt Fenstergröße fest`
  - `setTitle(String title); // Legt Fenstertitel fest`
- Daher am besten von `JFrame` erben:

```
// Beispielcode:
import javax.swing.JFrame;

public class Gui extends JFrame{

   public Gui(){
      this.setTitle(„Mein Fenster");
      this.setSize(300, 300);

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



Komponenten in Swing

Als Komponenten stehen sämtliche Bedienelemente zur Verfügung:

- `JButton, JTextField, JLabel`, usw.
- Aber auch neue `JPanel` (Füllwänd)

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class Gui extends JFrame{

   private JButton ok_btn;
   private JTextField txt_field;

   public Gui(){

      this.setTitle("Mein Fenster");
      this.setSize(300, 300);

      this.ok_btn = new JButton("OK");
      this.txt_field = new JTextField();

      this.add(this.txt_field);
      this.add(this.ok_btn);

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

Die Komponenten überlagern sich.
=> Wir brauchen also ein Layout

Layouts

Um die Elemente auf eine bestimmte Art anzuordnen, brauchen wir ein Layout.

- Swing hält verschiedene Layouts bereit (Auszug):
  - `FlowLayout ->` Anordnung von links nach rechts und oben nach unten
  - `GridLayout ->` Anordnung in Raster
  - `BorderLayout ->` Anordnung oben, unten, rechts, links, mitte
  - `BoxLayout ->` Anordnung in einzelner Zeile oder Spalte
- Einen guten Überblick über die verschiedenen Layouts gibt das Oracle Java Tutorial: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Hier als Beispiel:

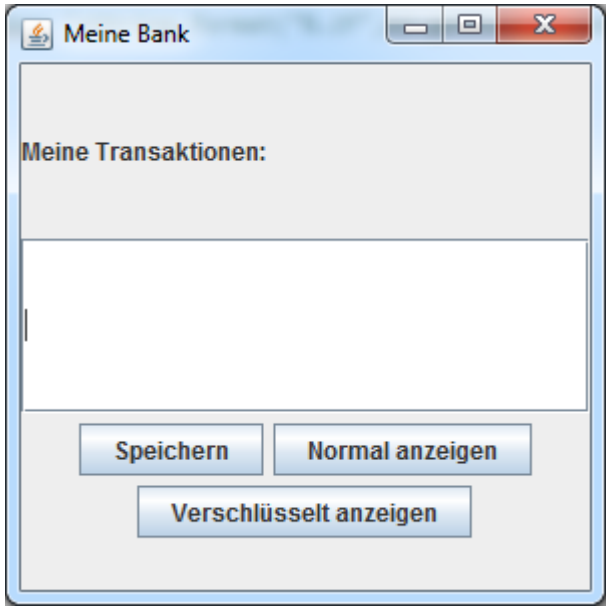
```
…
this.setLayout(new GridLayout(2, 1));
this.add(this.txt_field);
this.add(this.ok_btn);
```

Programmieraufgabe: GUI

Implementieren Sie nun für Ihre Bankkonto-Anwendung eine Einfache GUI,

- die ein Textfeld für die Ein- und Auszahlungen bereithält
- Zudem einen Button „Speichern“
- Und 2 Button „Normal anzeigen“ „Entschlüsselt anzeigen“

Ihre GUI kann beispielsweise wie folgt aussehen:



Auf Ereignisse reagieren

Beim Arbeiten mit GUIs interagiert der Benutzer mit versch. Komponenten

- Bewegt Maus, klickt Schaltfläche, verschiebt Rollbalken, …
- Das Programm muss darauf in geeigneter Weise reagieren

Es gibt Ereignis-Auslöser (event sources) wie z.B. Schaltflächen und es gibt Interessenten (Listener) für ein Event

- Interessenten melden sich bei einer Ereignisquelle an, um im Falle des Events benachrichtigt zu werden
- Ein Event wird dann ggf. an den registrierten Listener weitergeleitet
- Typische Listener sind:
  - ActionListener -> Benutzer aktiviert eine Schaltfläche oder ein Menü
  - WindowListener -> Benutzer schließt ein Fenster oder möchte es verkleinern
  - MouseListener –> Benutzer drückt auf eine Maustaste
  - MouseMotionListener -> Benutzer bewegt die Maus

Auf Ereignisse reagieren

Damit wir nun auf Events reagieren können müssen wir zwei Dinge tun:

1. Einen geeigneten Listener implementieren
2. Den Listener registrieren

Implementierung:

- Für jeden Listener gibt es ein Interface, das bestimmte Callback-Methoden vorschreibt
- In der Implementierung müssen wir festlegen, wie auf die einzelnen Events reagiert werden soll

Registrierung:

- Der Listener kann über den Aufruf von `addEreignisListener(EreignisListener)` zur einem bestimmten Ereignisauslöser hinzugefügt werden

Beispiel: ActionListener

Um nun auf Button-Klicks reagieren zu können, müssen wir das Interface `ActionListener` aus dem Packet java.awt.event implementieren:

- Dazu kann man entweder eine eigene Klasse anlegen:

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

public class MyActionListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        JOptionPane.showMessageDialog(null, "Sie haben Speichern gedrückt?" );

    }
}
```

Beispiel: ActionListener

Noch einfacher geht das aber entweder über

- Die Verwendung von anonymen inneren Klassen
- Oder seit Java 8 auch über Lambda-Ausdrücke

```
// Einfaches Beispiel:

    this.txt_field = new JTextField();
    this.ok_btn = new JButton("OK");
    this.ok_btn.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {

        txt_field.setText("OK");
      }
    });

// Per Lambda-Ausdruck:
this.ok_btn.addActionListener((ActionEvent e)-> txt_field.setText("OK"));
```

Hausaufgabe

Versuchen Sie nun Ihre eben erstellte GUI mit Leben zu füllen, indem Sie beim Klick des entsprechenden Buttons die folgenden Aktionen durchführen:

- Speichern: Speichert den Text im Textfeld in eine Datei
- Normal anzeigen: Zeigt den Text von der Datei im Klartext an
- Verschlüsselt anzeigen: Zeigt den Text aus der Datei verschlüsselt an
  - Nehmen Sie hierzu den FilterReader aus der vorherigen Aufgabe