LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Fortgeschrittene

Einheit 04: Einführung in JavaFX

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

Einführung in JavaFX

- Motivation und Eigenschaften
- Hello World in JavaFX
- Komponenten und Szenegraph
- Nutzeraktionen

Praxis:

- Log-In Fenster entwerfen
- Log-In Fenster: Button Logik
- Hausaufgabe: Bankanwendung in JavaFX schreiben und einbinden

Lernziele

- Grundlagen in JavaFX kennenlernen
- GUIs erstellen und verwenden können
- Konzepte von GUI-Programmierung in Java verstehen

Motivation

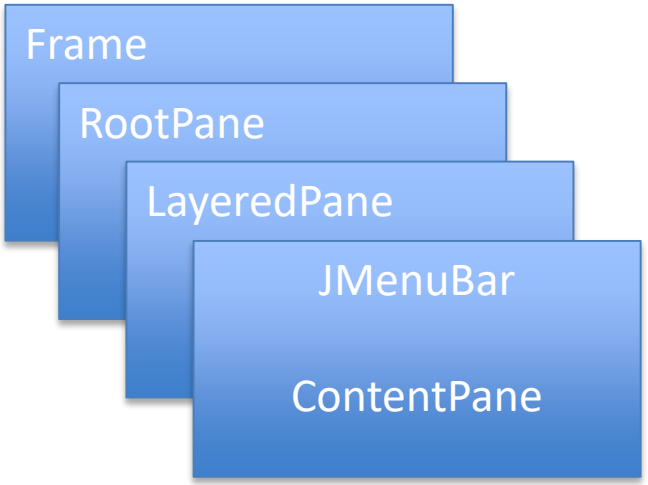
GUI – **G**raphical **U**ser **I**nterface

- Anschaulichere und leichtere Möglichkeit zur Dateneingabe und Kontrolle von Programmen
- JavaFX:
  - Große und vielseitige Bibliothek zur Gestaltung grafischer Oberflächen
  - Zur Entwicklung von Desktop und RIAs (Rich Internet Applications)
  - Seit Java SE 7 Update 6 als fester Bestandteil von Java SE x86
    - Dez. 2008: JavaFX 1.0 von Sun freigegeben
    - Mitte 2011: JavaFX 2.0 – auf JavaFX Script wird verzichtet
    - Heute: JavaFX 8: Entwickelt von Oracle
  - Soll Swing ablösen
  - Schnell erstellbare neue UI-Komponenten (per CSS gestaltbar)
  - JavaFX Anwendungen auf nahezu allen Geräten ausführbar
    - Auch im Browser
  - Grafische WYSIWYG Tools: JavaFX Scene Builder

Eigenschaften

Zur Wiederholung: Swing

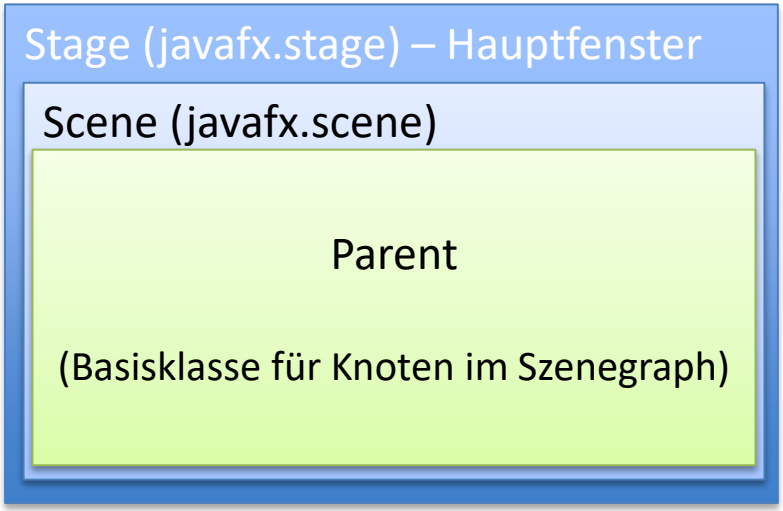
- `JFrame` beinhaltet das `JRootPane` als einziges Kind
  - Stellt die `ContentPane` zur Verfügung
    - Ist die Basis-Komponente für alle Unterkomponenten



In JavaFX arbeiten wir mit Szenegraphen im Hauptfenster:

Die wichtigsten Komponenten:

- Hauptfenster ist unsere Bühne (Stage)
- Auf der Bühne gibt es Szenen
- Die Szene besitzt die Elemente in einer baumartigen Struktur, dem Szenegraphen



Hello World! in JavaFX


```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override public void start(Stage stage) {
        Text text = new Text(10, 40, "Hello World!");
        text.setFont(new Font(40));
        Scene scene = new Scene(new Group(text));

        stage.setTitle("Welcome to JavaFX!");
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }

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

Hauptklasse einer JavaFX Applikation erbt von javafx.application.Application

Einstiegspunkt für jede JavaFX Anwendung

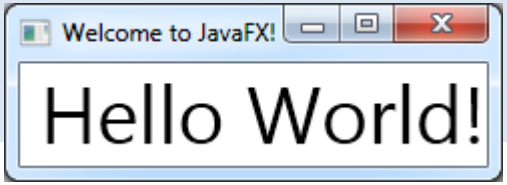
Bekommt die Bühne

Die Szene mit dem Text errichten

Die Szene auf die Bühne holen

Die Bühne anzeigen

Klassische Main-Methode: Für JAR-Files ohne JavaFX Launcher (Optional)



Komponenten im Detail: `Stage`

Hauptfenster (`Stage`) aus `javafx.stage`

- `public class Stage extends Window`
- Stellt den obersten JavaFX Container bereit
- Die Hauptbühne wird von der Plattform erzeugt und beim Start als Argument übergeben, häufig: `public void start(Stage primaryStage){…}`
- Zusätzliche Bühnen können von der Anwendung (Application Thread) erzeugt und verändert werden.
- Kann Eigenschaften des Hauptfensters anzeigen oder festlegen, wie bsp.:
  - `void setTitle(String title); // Titel des Fensters festlegen`
  - `void setMaxWidth(double value);// Maximale Breite festlegen`
  - `Boolean isMaximized(); // Ist das Fenster maximiert?`
- Kann aber nicht direkt Elemente aufnehmen, sondern braucht eine Szene:
  - `primaryStage.setScene(meineSzene);`

Komponenten im Detail: Scene

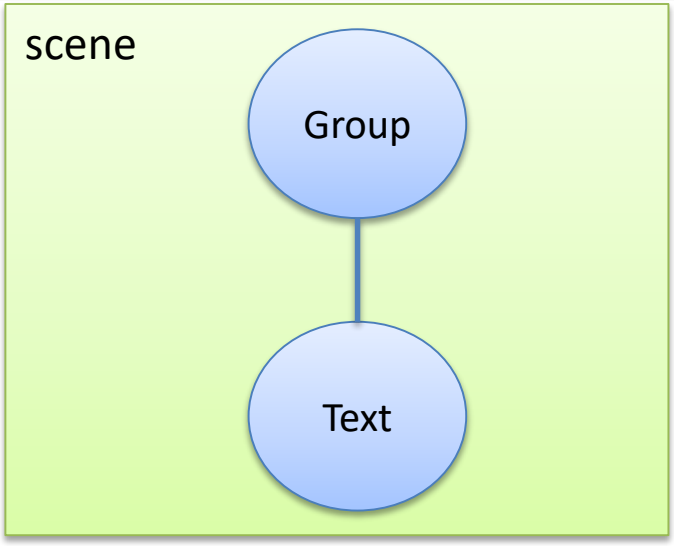
Szene (Scene) aus `javafx.scene`

- `public class Scene`
- Stellt den Container für alle Inhalte des Szenegraph bereit
  - Dazu muss ein Wurzelknoten (Root-Node) vom Typ `Parent` angegeben werden
- Ausgehend von diesem Wurzelknoten können nun Elemente hinzugefügt werden:
  - Meist verwendet man als `Parent root` eine Glasscheibe (`Pane`) auch für das Layout (`javafx.scene.layout.Pane`), die wiederum mehrere Objekt enthalten können - auch weitere `Panes`
  - So entsteht der Szenegraph als eine Art Baum, dessen Blätter graphische Elemente (Buttons, Textfelder, usw.) und die Verzweigungen weitere `Panes` sind.
    - `Scene scene = new Scene(new Group(text));`

Constructor Summary

Constructors

Constructor and Description
`Scene(Parent root)` Creates a Scene for a specific root Node.
`Scene(Parent root, double width, double height)` Creates a Scene for a specific root Node with a specific size.
`Scene(Parent root, double width, double height, boolean depthBuffer)` Constructs a scene consisting of a root, with a dimension of width and height, and sp
`Scene(Parent root, double width, double height, Paint fill)` Creates a Scene for a specific root Node with a specific size and fill.
`Scene(Parent root, Paint fill)` Creates a Scene for a specific root Node with a fill.



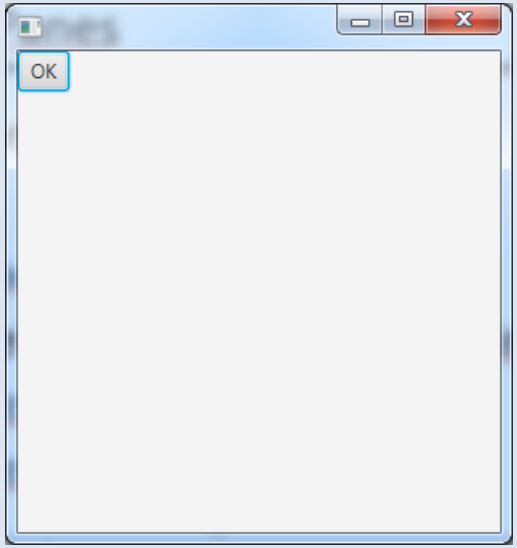
Einfacher Szenegraph aus HelloWorld

Arbeiten mit Panes

Wir können mit Glasscheiben sehr einfach arbeiten:

- Erstellen:
  - `Pane glasscheibe = new Pane();`
- Elemente Hinzufügen (Indirekt über die Liste der Kindknoten):
  - `glasscheibe.getChildren().add(text);`
  - `glasscheibe.getChildren().add(button);`

```
@Override public void start(Stage stage) {

      // Elemente
      Text text = new Text("Hello World!");
      Button button = new Button("OK");

      // Glasscheibe
      Pane glasscheibe = new Pane();

      // Elemente auf die Glasscheibe stellen
      glasscheibe.getChildren().add(text);
      glasscheibe.getChildren().add(button);

      //Scene erstellen
      Scene szene = new Scene(glasscheibe,300,300);

       // Die Szene auf die Bühne holen
       stage.setScene(szene);
       stage.show();
}
```

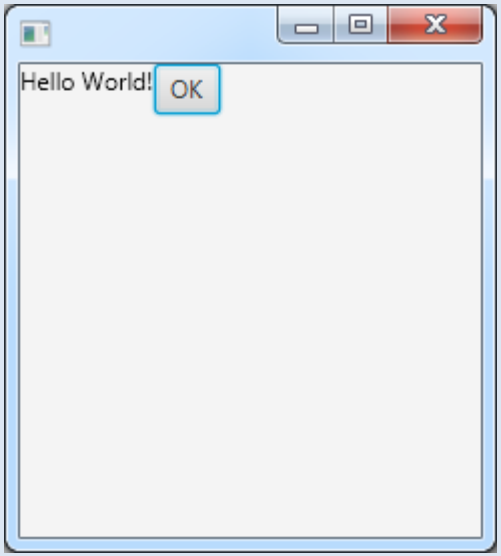


Es wurde kein Layout spezifiziert:
Daher werden alle Elemente übereinander gelegt.

Layouts können direkt mit einer Glasscheibe verknüpft sein: *Built-in Layout Panes* (`javafx.scene.layout.Pane`)

- Guter Überblick unter: https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
  - Bsp.: `AnchorPane, BorderPane, FlowPane, GridPane, HBox, StackPane, etc.`

```
@Override public void start(Stage stage) {

    // Elemente
    Text text = new Text("Hello World!");
    Button button = new Button("OK");

    // Glasscheibe mit HBox Layout:
    HBox box = new HBox();

    // Elemente auf die Glasscheibe stellen
    box.getChildren().add(text);
    box.getChildren().add(button);

    //Scene erstellen
    Scene szene = new Scene(box,300,300);

       // Die Szene auf die Bühne holen
       stage.setScene(szene);
       stage.show();
  }
```

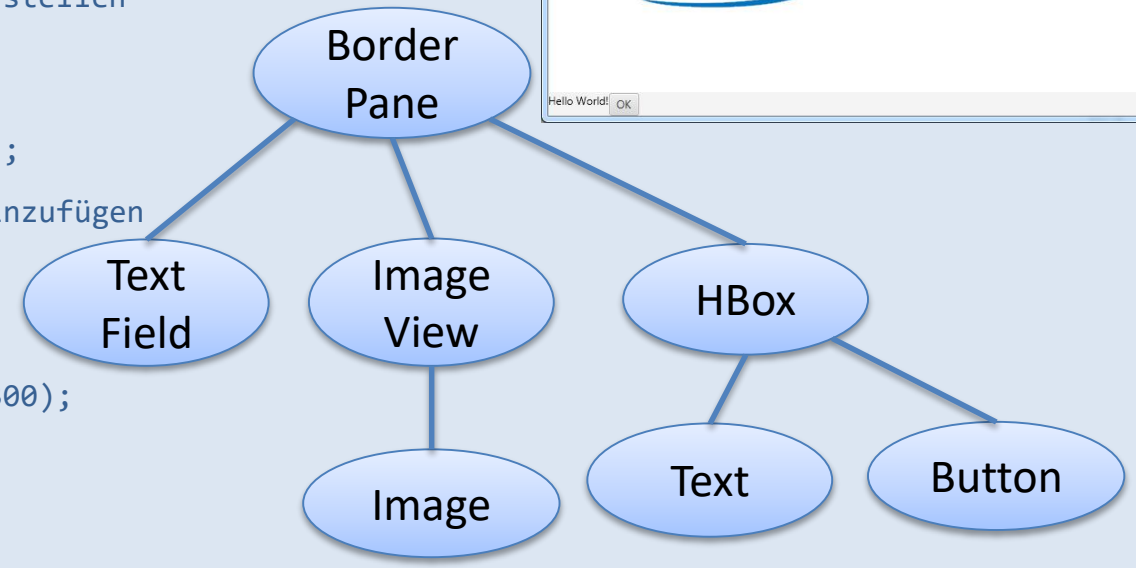


Erweiterter Szenegraph

Panes können Elemente und auch wieder Panes mit Elementen enthalten:

- Bsp.: eine BorderPane hat eine HBox Pane und andere Elemente:

```
@Override public void start(Stage stage) {

      // Elemente
      Text text = new Text("Hello World!");
      Button button = new Button("OK");
      TextField txt_field = new TextField();
      txt_field.setMinWidth(20);
      ImageView iv = new ImageView(new Image("java.png"));

      // Glasscheibe mit HBox Layout:
      HBox box = new HBox();

      // Elemente auf die Glasscheibe stellen
      box.getChildren().add(text);
      box.getChildren().add(button);

      // Glasscheibe mit BorderLayout:
      BorderPane bp = new BorderPane();

      // Glasscheibe zur Borderpane hinzufügen
      bp.setTop(txt_field);
      bp.setCenter(iv);
      bp.setBottom(box);

      //Scene erstellen
      Scene szene = new Scene(bp,300,300);

       //…
   }
```

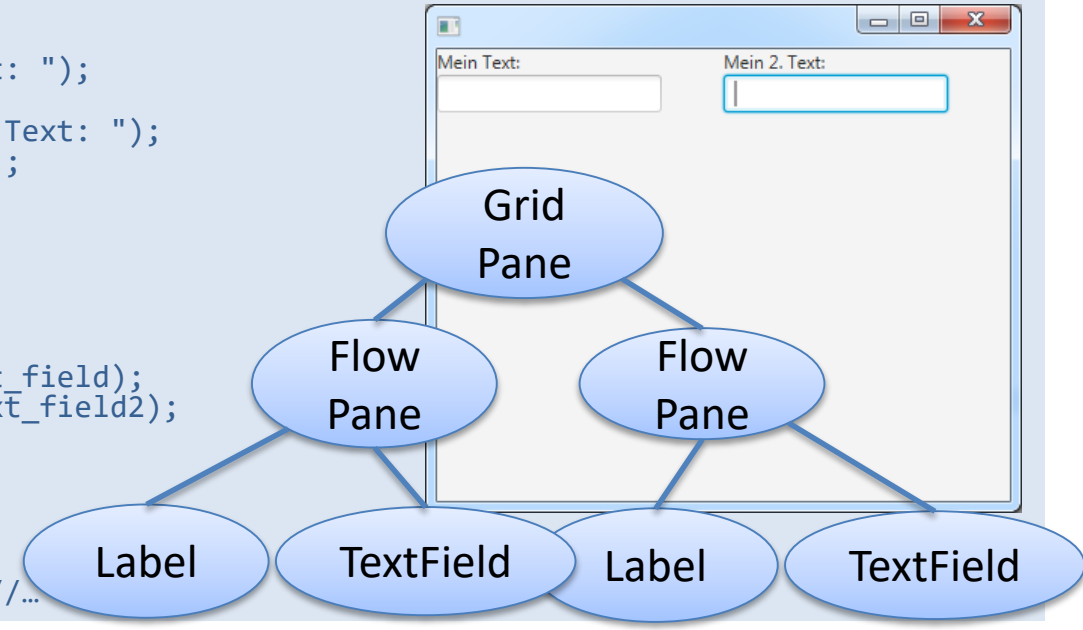


Umgang mit Komponenten

Ein paar Hinweise im Umgang mit Komponenten:

- Aktive Komponenten (bspw.: Textfelder, Buttons, etc.) sollten als Instanzvariablen deklariert werden, um sie später benutzen zu können.
  - Bsp.: `private Button ok_btn;`
- Logisch zusammengehörige Komponenten sollten i.d.R. auch in einer Pane vereinigt werden. Bsp.:
  - Textfelder und deren Beschriftungen (Labels)
  - Buttons (Ok und Cancel)

```
// Beispiel für Labels und Textfelder in einer FlowPane:

      // Elemente
      Label txt_label = new Label("Mein Text: ");
      TextField txt_field = new TextField();
      Label txt_label2 = new Label("Mein 2. Text: ");
      TextField txt_field2 = new TextField();

      // Flow Layout:
      FlowPane fp1 = new FlowPane();
      FlowPane fp2 = new FlowPane();

      // Elemente hinzufügen
      fp1.getChildren().addAll(txt_label,txt_field);
      fp2.getChildren().addAll(txt_label2,txt_field2);

      GridPane gp = new GridPane();
      gp.add(fp1, 0, 0);
      gp.add(fp2,0,1);

      Scene szene = new Scene(gp,300,300); //…
```



Aufgabe zu JavaFX

Erstellen Sie ein JavaFX Projekt „HelloFX“ in Eclipse und erstellen Sie eine GUI:

- Mit einer großen Überschrift: „Willkommen bei Ihrer Bank!“
- Mit einem Textfeld für den Nutzername
- Mit einem Passwortfeld für das Passwort
- Und 1 Button „Einloggen“

Die zu erstellende GUI könnte in etwa so aussehen:

Auf Nutzeraktionen reagieren

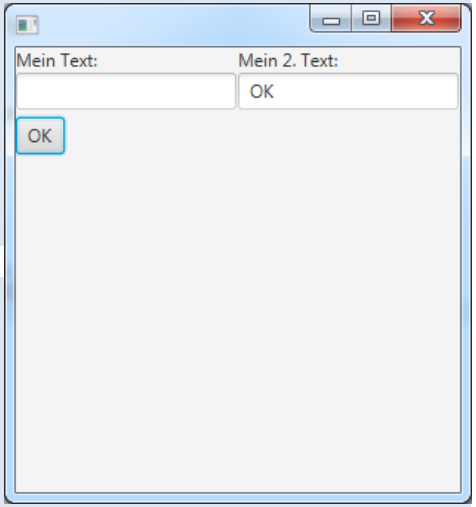
Wir wollen nun auch auf Nutzeraktionen reagieren:

- TextFelder können per `setText(String text)` und `getText()` verändert bzw. ausgelesen werden.

- Um auf Button-Klicks reagieren zu können müssen wir dem Button mittels `setOnAction` einen action handler (Behandler) anheften.
  - Reagiert immer dann, wenn der Button gedrückt wurde
  - Implementiert das funktionale Interface `EventHandler<ActionEvent>`, welches die Methode `public void handle(ActionEvent e)` bereitstellt
  - Sehr ähnlich zu Swing:
    - mit `ActionListener` und `public void actionPerformed(ActionEvent e)`
  - Das geht sehr einfach über eine anonyme innere Klasse
  - Oder seit Java 8 über Lambda-Ausdrücke

setOnAction


```
//Einfaches Beispiel mit anonymer innerer Klasse:

//…
   TextField txt_field2 = new TextField();

   Button mein_button = new Button("OK");
   mein_button.setOnAction(new EventHandler<ActionEvent>() {

      @Override
      public void handle(ActionEvent event) {
         txt_field2.setText("OK");
      }
   });
```




```
//Einfaches Beispiel mit Lambda-Ausdrücken (Seit Java 8)

//…
   Label txt_label2 = new Label("Mein 2. Text: ");
   TextField txt_field2 = new TextField();

   Button mein_button = new Button("OK");
   mein_button.setOnAction(event -> txt_field2.setText("OK"));
```

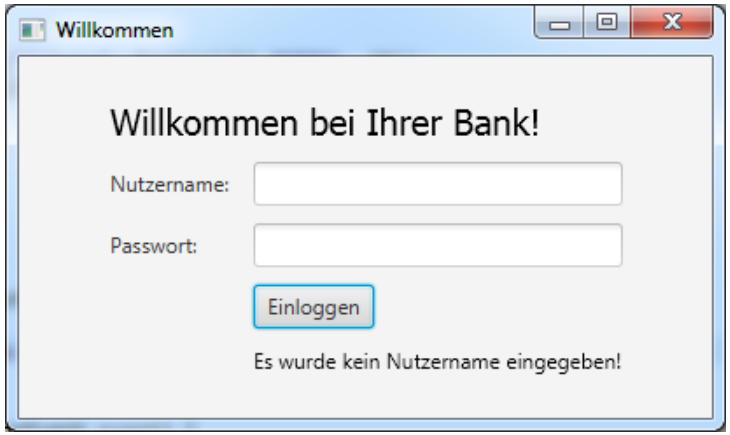
Aufgabe zu JavaFX: Button mit Leben füllen

Füllen Sie nun Ihre zuvor erstellte GUI mit Leben, in dem Sie auf den Einloggen-Button reagieren.

Erstellen Sie zunächst einen Platzhalter für einen Status-Text unterhalb des Einloggen-Buttons.

Reagieren Sie nun wie folgt, wenn der Einloggen-Button gedrückt wurde:

- Falls kein Nutzername eingegeben wurde, erscheint als Status-Text: „Es wurde kein Nutzername eingegeben!“
- Falls kein Passwort Eingegeben wurde, erscheint als Status-Text: „Es wurde kein Passwort eingegeben!“
- Falls Nutzername oder Passwort nicht mit einem von Ihnen akzeptierten Nutzername oder Passwort übereinstimmt, dann geben Sie aus: „Nutzername oder Passwort falsch!“
- Falls beides korrekt, dann geben Sie aus: „Nutzer wird eingeloggt.“



Neuer Inhalt im gleichen Fenster

Will man nun den Inhalt in seinem Fenster neu gestalten, bspw. wenn der Nutzer eingeloggt ist, wird einfach eine neue Szene eingesetzt:

- `primaryStage.setScene(new Scene(new NeueSzene()));`
- Kann als separate Klasse Definiert werden
  - Muss von `Parent` erben

Hausaufgabe:

Nachdem Sie nun bereits ein funktionsfähiges Log-In Fenster gestaltet haben, sollten Sie nun eine Szene in JavaFX für die Bank-Anwendung von letzter Stunde schreiben und bei einem erfolgreichen Log-In diese aufrufen.

Die Funktionen der Buttons sollten dann genauso funktionieren, wie bei der letzten Hausaufgabe gefordert.

Zum Vergleich, dies war die
GUI für die Bankanweisung in Swing =>