



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



 mobile and
distributed systems group



Javakurs für Fortgeschrittene

Einheit 04: JavaFX

Kyrill Schmid

Lehrstuhl für Mobile und Verteilte Systeme



Einführung in JavaFX

- Motivation und Eigenschaften
- Hello World in JavaFX
- Komponenten und Szenegraph
- Nutzeraktionen
- GUI gestalten mit CSS

Praxis:

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

Lernziele

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

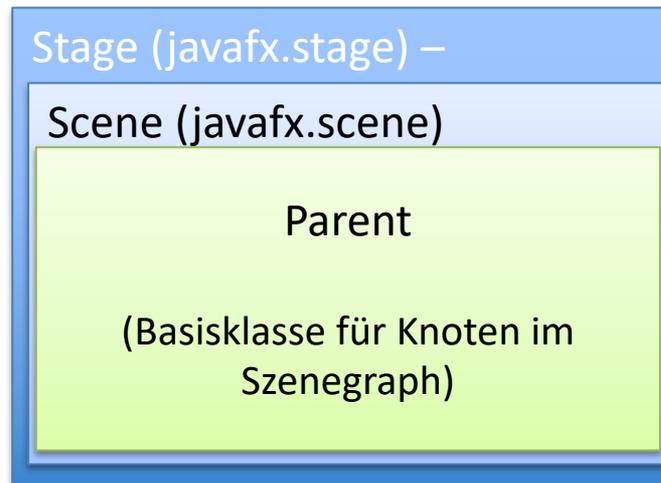
GUI – Graphical User Interface

- Anschauliche und leichtere Möglichkeit zur Dateneingabe und Kontrolle von Programmen
- JavaFX:
 - Große und vielseitige Bibliothek zur Gestaltung grafischer Oberflächen
 - War seit JDK 7 in das JDK integriert
 - Wird aber seit Java 11 unabhängig vom JDK entwickelt und veröffentlicht
 - Zur Entwicklung von Desktop und RIAs (Rich Internet Applications)
 - Schnell erstellbare neue UI-Komponenten (per CSS gestaltbar)
 - JavaFX Anwendungen auf nahezu allen Geräten ausführbar
 - Auch im Browser
 - Grafische WYSIWYG Tools: JavaFX Scene Builder



In JavaFX arbeiten wir mit Szenegraphen im Hauptfenster. Die wichtigsten Komponenten:

1. Hauptfenster ist unsere Bühne (**Stage**)
2. Auf der Bühne gibt es Szenen (**Scene**)
3. Die Szene besitzt die Elemente in einer baumartigen Struktur, dem Szenegraph (**Nodes**)



```
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

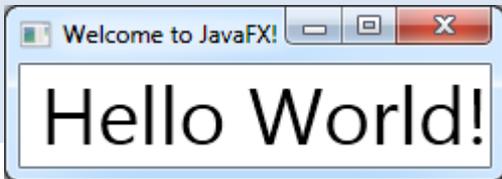
Bekommt die Bühne

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)



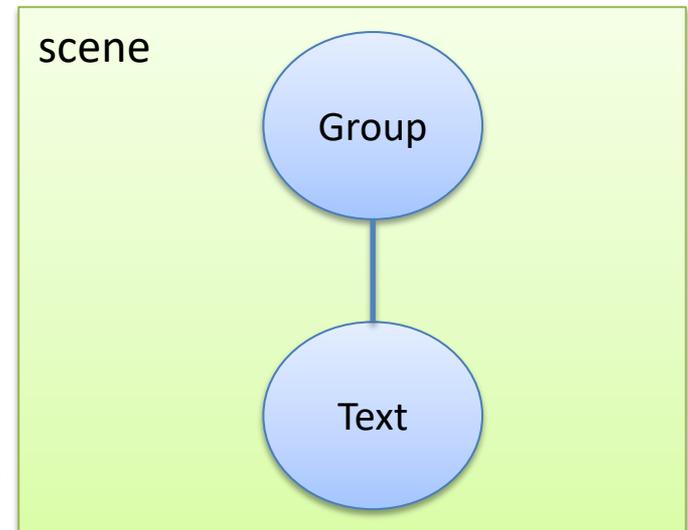
Hauptfenster (Stage) aus `javafx.stage`

- `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);`

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:
 - `public Scene(Parent root);` // Ein Konstruktor von Scene
 - 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));`

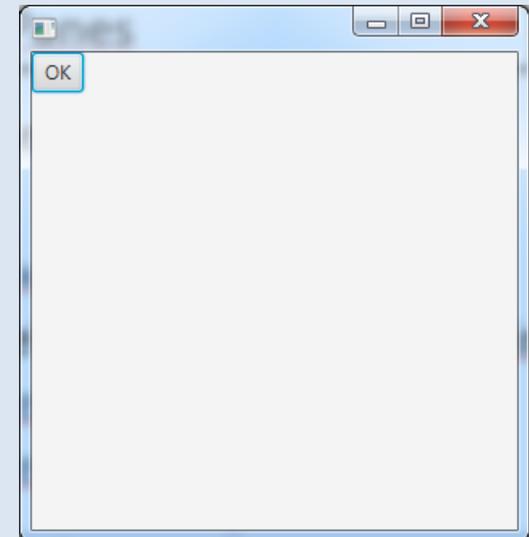
Einfacher Szenegraph
aus HelloWorld



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 scene = new Scene(glasscheibe,300,300);  
  
    // Die Szene auf die Bühne holen  
    stage.setScene(scene);  
    stage.show();  
}
```



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

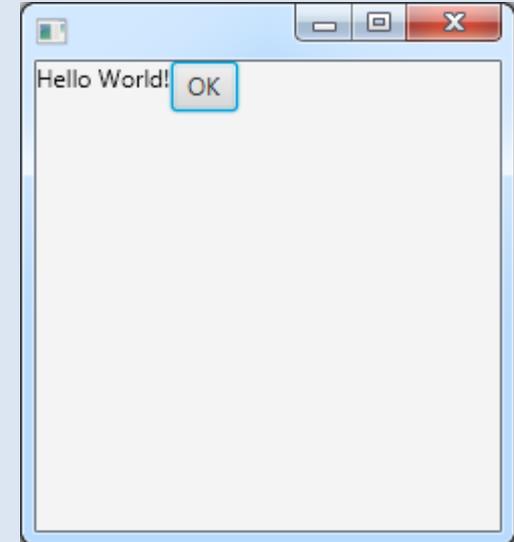
Layouts können direkt mit einer Glasscheibe verknüpft sein: Built-in Layout Panes (`javafx.scene.layout.Pane`)

- 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 scene = new Scene(box,300,300);  
  
    // Die Szene auf die Bühne holen  
    stage.setScene(scene);  
    stage.show();  
}
```



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();

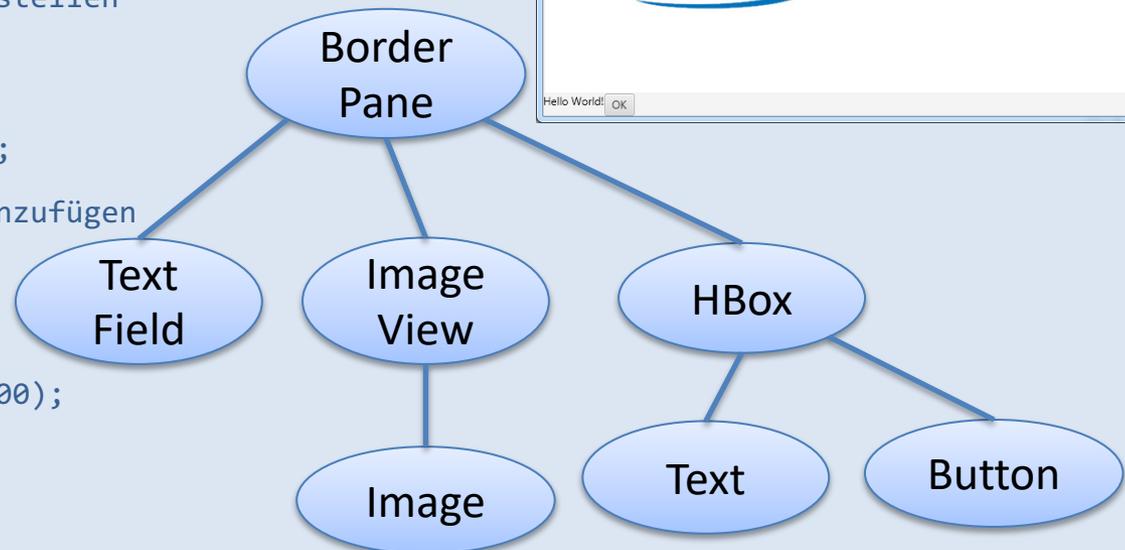
    // Elemente auf die Glasscheibe stellen
    box.getChildren().add(text);
    box.getChildren().add(button);

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

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

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

    //...
}
```



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:
```

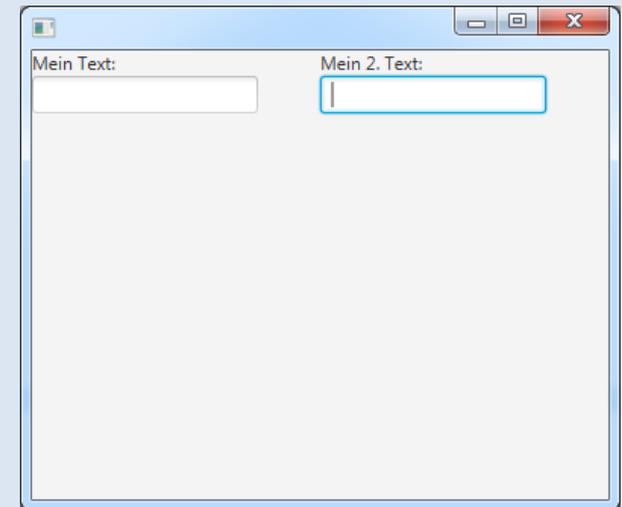
```
// Elemente
Label txt_label = new Label("Mein Text: ");
TextField txt_field = new TextField();
Label txt_label2 = new Label("Mein 2. Text: ");
TextField txt_field2 = new TextField();

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

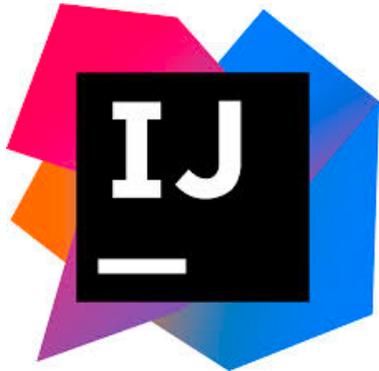
// Elemente hinzufügen
fp1.getChildren().addAll(txt_label,txt_field);
fp2.getChildren().addAll(txt_label2,txt_field2);

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

Scene scene = new Scene(gp,300,300); //...
```



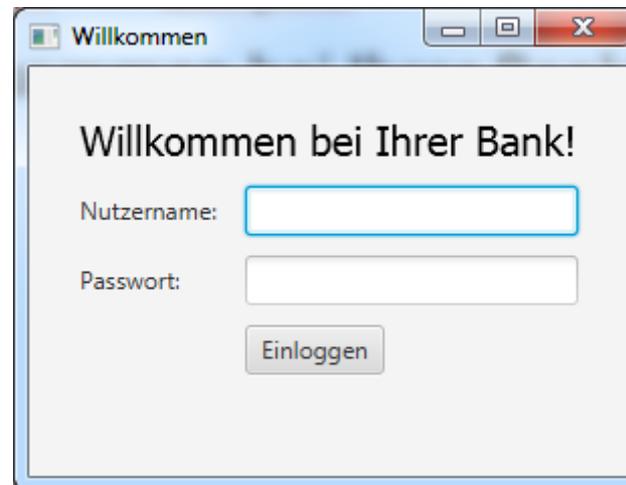
- Unter: <https://gluonhq.com/products/javafx/> die passende Version herunterladen:
- Anleitung für für verschiedene IDEs: <https://openjfx.io/openjfx-docs/#install-javafx>



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 Nutzernamen
- Mit einem Passwortfeld für das Passwort
- Und 1 Button „Einloggen“

Die zu erstellende GUI könnte in etwa so aussehen:



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

Anonyme Klassen haben

- Keinen Namen
- Erzeugen bei der Klassendeklaration automatisch ein Objekt
- Klassendeklaration und Objekterzeugung sind so zu einem Sprachkonstrukt verbunden

```
// Allgemeine Syntax
```

```
new KlasseOderSchnittstelle() { /* Eigenschaften der inneren Klasse */ }
```

Zwei Möglichkeiten

- Steht hinter `new` ein Klassentyp, dann ist die anonyme Klasse eine Unterklasse dieser Klasse
- Steht hinter `new` der Name einer Schnittstelle, dann erbt die anonyme Klasse von `Object` und implementiert die Schnittstelle

Weitere Eigenschaften anonymer innerer Klassen

- Können keine weiteren Schnittstellen implementieren oder von weiteren Klassen erben
- Erlauben keine eigenen Konstruktoren
- Nur Objektmethoden und finale statische Variablen erlaubt

```
// Beispiel
public class HelloWorld {
    public void greet() {
        System.out.println("Hello world!");}
}
...
public static void main(String... args) {

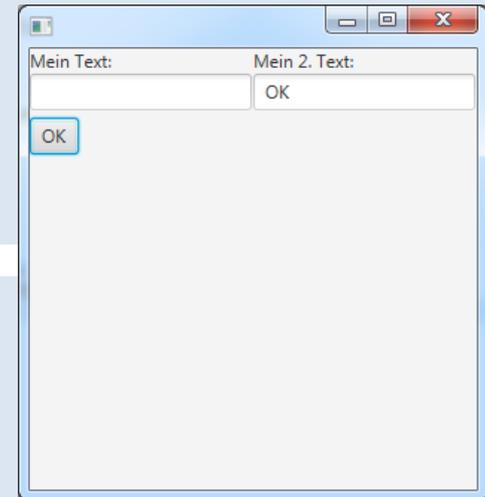
    HelloWorld germanGreeting = new HelloWorld() {
        public void greet() {
            System.out.println("Hallo Welt!" ); }};
    germanGreeting.greet();
}
```

//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();

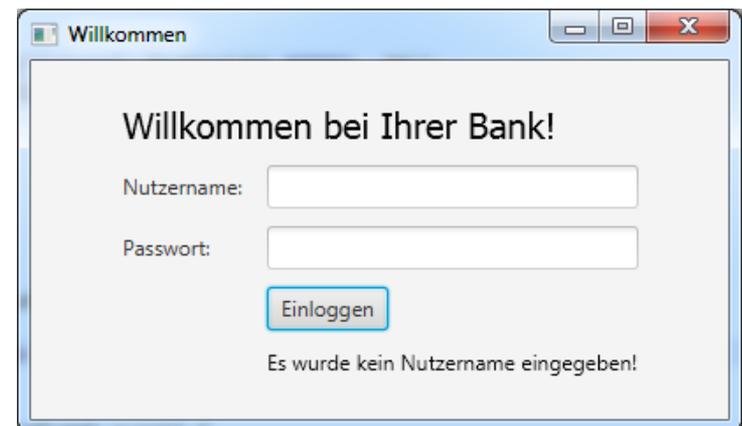
Button mein_button = new Button("OK");
mein_button.setOnAction(event -> txt_field2.setText("OK"));
```

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



Bisher haben wir mit Standard Design-Elementen gearbeitet!

In Swing mussten die Elemente i.d.R. einzeln designed werden:

- `myButton.setBackground(new Color...);`
- `myButton.setBorder(...);`

Oder man nutzte open-source Frameworks

(Javaxx, Java CSS,...) um cascading style sheets (CSS) mit Swing Komponenten zu verwenden.

In JavaFX ist die Trennung von Inhalt und Layout fest verankert und wird mit *JavaFX CSS* verwirklicht.

- Basiert auf W3C CSS 2.1 und folgt den bekannten Regeln für CSS
- Reference Guide unter:

<https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>



Ein paar Unterschiede zu CSS:

- JavaFX Eigenschaften werden mit dem Präfix `fx` erweitert
- verbietet CSS Layout-Eigenschaften, wie `float`, `position`, usw.
- bietet einige Erweiterungen, bspw.: (Hintergrund-)Farben, Ränder, usw.

CSS Styles werden nun für Knoten im JavaFX Szenegraphen verwendet.

- Für das Mapping gelten die bekannten CSS Regeln für Selektoren:
 - **Typ-Selektoren:**
 - Analog zu einem Element in HTML
 - I.d.R. einfach der Name der Klasse, bspw.: `Button`, `Text`, usw.
 - Kann abgefragt werden mittels `public String getTypeSelector()`

```
// Beispiel für Typ-Selektor Definition:
```

```
Button {  
-fx-font-size: 25px;  
-fx-font-weight: bold;  
-fx-font-style: italic;  
-fx-font-family: "Arial Blank"  
}
```



▪ **Klassen-Selektoren:**

- Jeder Knoten im Szenegraph kann zu einer oder mehreren Klassen gehören (analog zum class-Attribut in HTML)
- Elemente können mittels `getStyleClass().add(String class)` zu einer Klasse hinzugefügt werden:
 - Bsp.: `myButton.getStyleClass().add("MeineKlasse");`
- Ansprechbar dann mit bekannter Punkt-Notation:

```
.MeineKlasse{  
    -fx-font-weight: bold;  
}
```

▪ **ID- Selektoren:**

- Jeder Knoten besitzt ein ID-Attribut (analog zu id in HTML)
 - Kann mit `setID(String id)` gesetzt werden.
 - Bsp.: `myButton.setId("button1");`
- Ansprechbar dann mit bekannter #-Notation

```
#button1{  
    -fx-font-weight: bold;  
}
```

Daneben stehen uns auch ein Teil der sog. Pseudoklassen zur Verfügung:

Pseudoklasse	Auswirkung
Focused	Wenn das Element den Fokus erhält
Hover	Wenn der Mouse-Zeiger über dem Element steht
Pressed	Wenn das Element angeklickt wird

```
// Beispiel für Pseudoklassen:
```

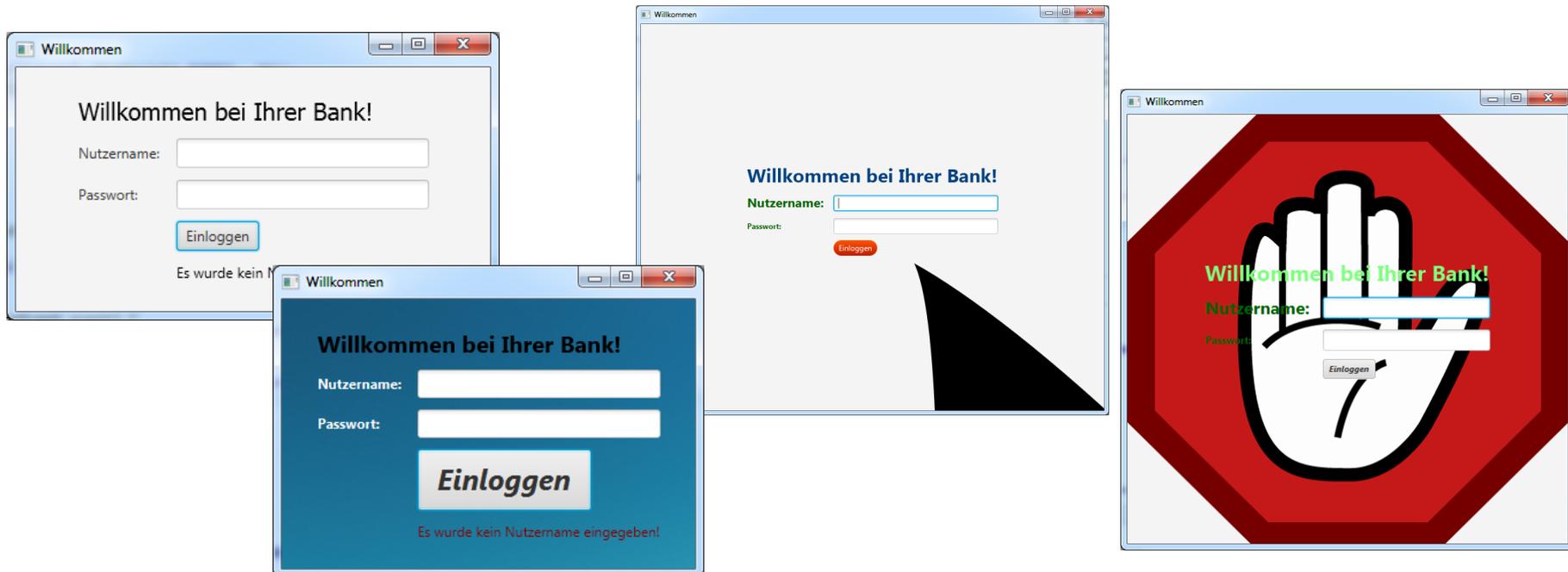
```
Button:hover {  
  
    -fx-border-width: 0.0 ;  
    -fx-font-size: 10px;  
    -fx-font-weight: bold;  
    -fx-font-style: italic;  
    -fx-font-family: "Arial Blank";  
  
}
```

2 Arten, um CSS in unsere JavaFX-Applikation einzubinden (analog zu HTML):

- Inline Styles
 - Direkt im Code (Class File)
 - Jeder `Node` verfügt über die Methode `public final void setStyle(String value)`
 - Bsp.: `myButton.setStyle("-fx-font-size: 20px");`
 - Hat höchste Priorität, wäre aber sehr umständlich
- Style sheets
 - Eigene separate CSS Datei.css
 - Wird i.d.R. der Szene mitgeteilt:
 - `scene.getStylesheets().add(getClass().getResource("application.css").toString());`
 - Damit können wir globale Design-Einstellungen für eine Szene definieren.
 - Sehr komfortabel

Verwenden Sie nun JavaFX CSS Elemente, um Ihre zuvor erstellte GUI individuell nach Ihren Bedürfnissen zu gestalten.

- Seien Sie dabei etwas kreativ und spielen ein wenig mit den Möglichkeiten, die Ihnen JavaFX CSS bietet.
- Verwenden Sie auch einmal inline Styles bzw. verschiedene Style Sheets.
- Ihre GUI könnte sich wie folgt verändern:



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

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 =>

