Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

V2 – GUI-Architekturen

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Um was geht es?

- Was sind GUI-Architekturen?
- Was ist ereignisorientierte Programmierung?
- Wie wird das Observer Design Pattern in GUI-Applikationen verwendet?
- Wie werden gängige Architektur-Patterns für GUI-Applikationen wie MVC, MVP und MVVM verwendet?

Lernziele LE 11 – GUI-Architekturen

Sie sind in der Lage,

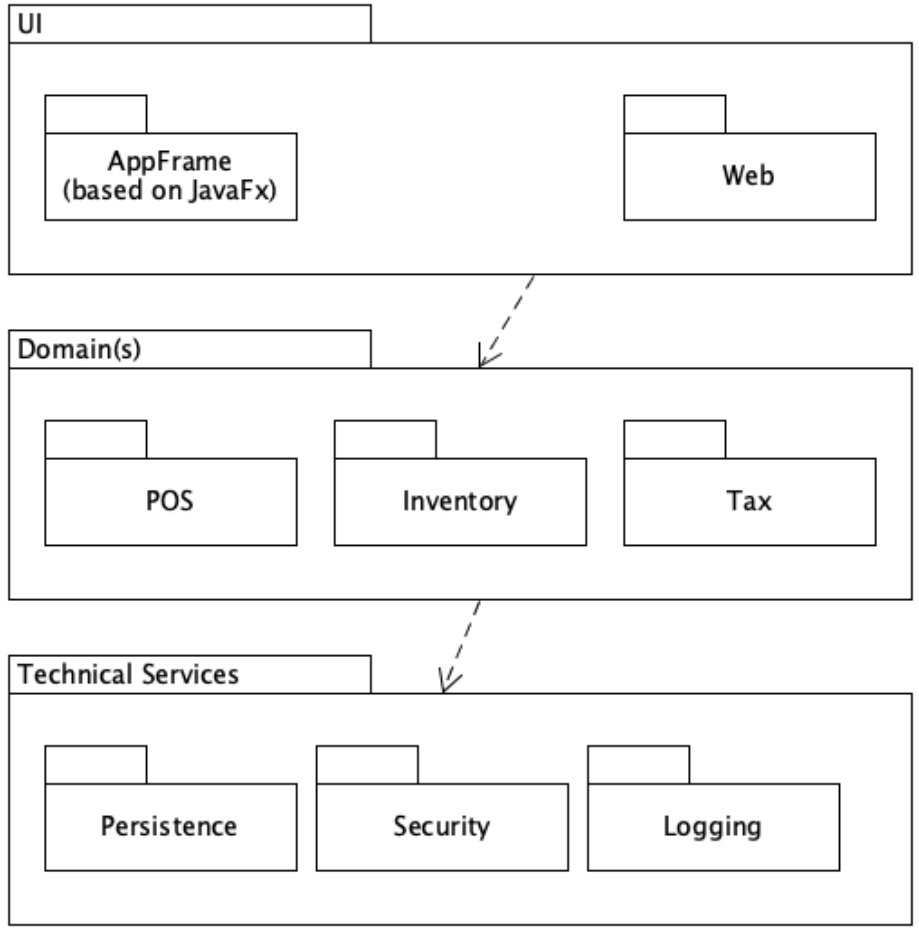
- zu erläutern, wie GUI-Architekturen entworfen und umgesetzt werden können,
- das Observer-Pattern in GUI-Applikationen einzusetzen,
- Beispiele für die Anwendung der Architektur-Patterns MVC, MVP und MVVM zu verstehen und zu erweitern,
- die Varianten für die ereignisorientierte Programmierung für JavaFX praktisch anzuwenden.

Agenda

1. **Einführung in GUI-Architekturen**
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

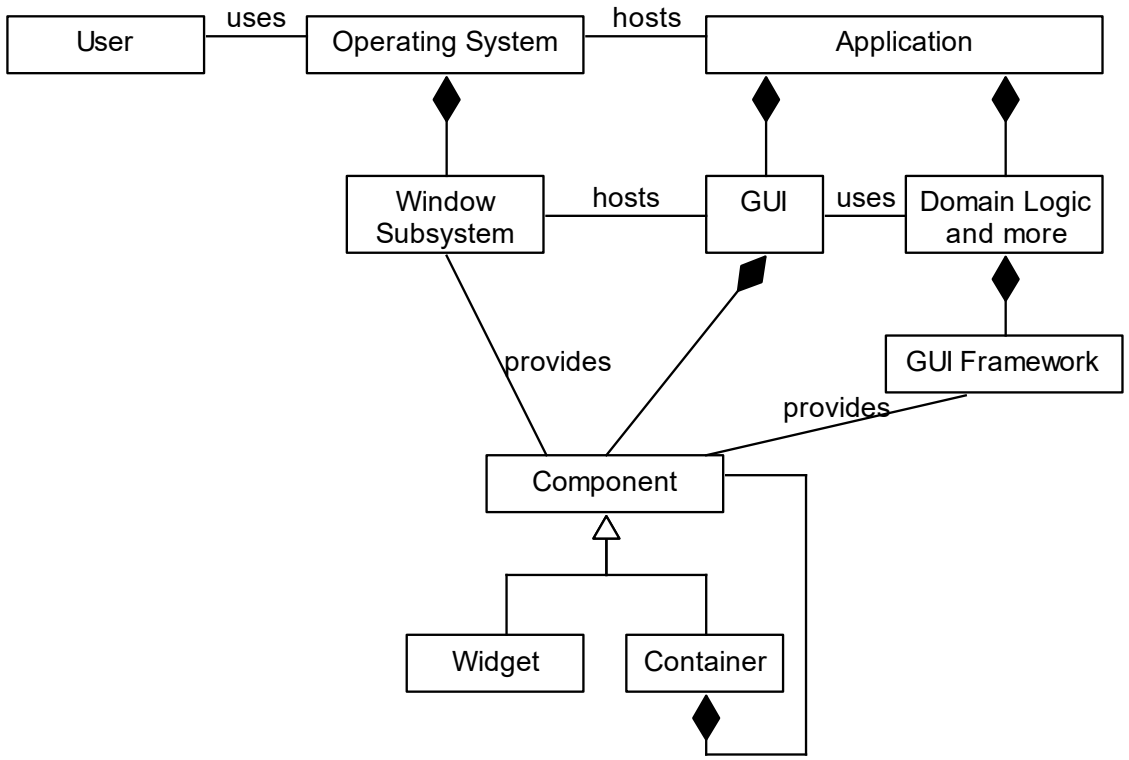
Recap - Schichtenarchitektur

- UI (Presentation): Enthält die Präsentations-Logik
  - Technologie-gebunden (rascher Wechsel)
  - Desktop-App (Java: Swing oder JavaFX)
  - Mobile-App (iOS, Android, Hybrid-Frameworks)
  - Web-App (HTML, CSS, JS, Frameworks)
- Domain: Enthält den fachlichen Kern der Anwendungslogik.
- Technical Services: Enthält in Abhängigkeit der technischen Infrastruktur wichtige technologieabhängige Komponenten.



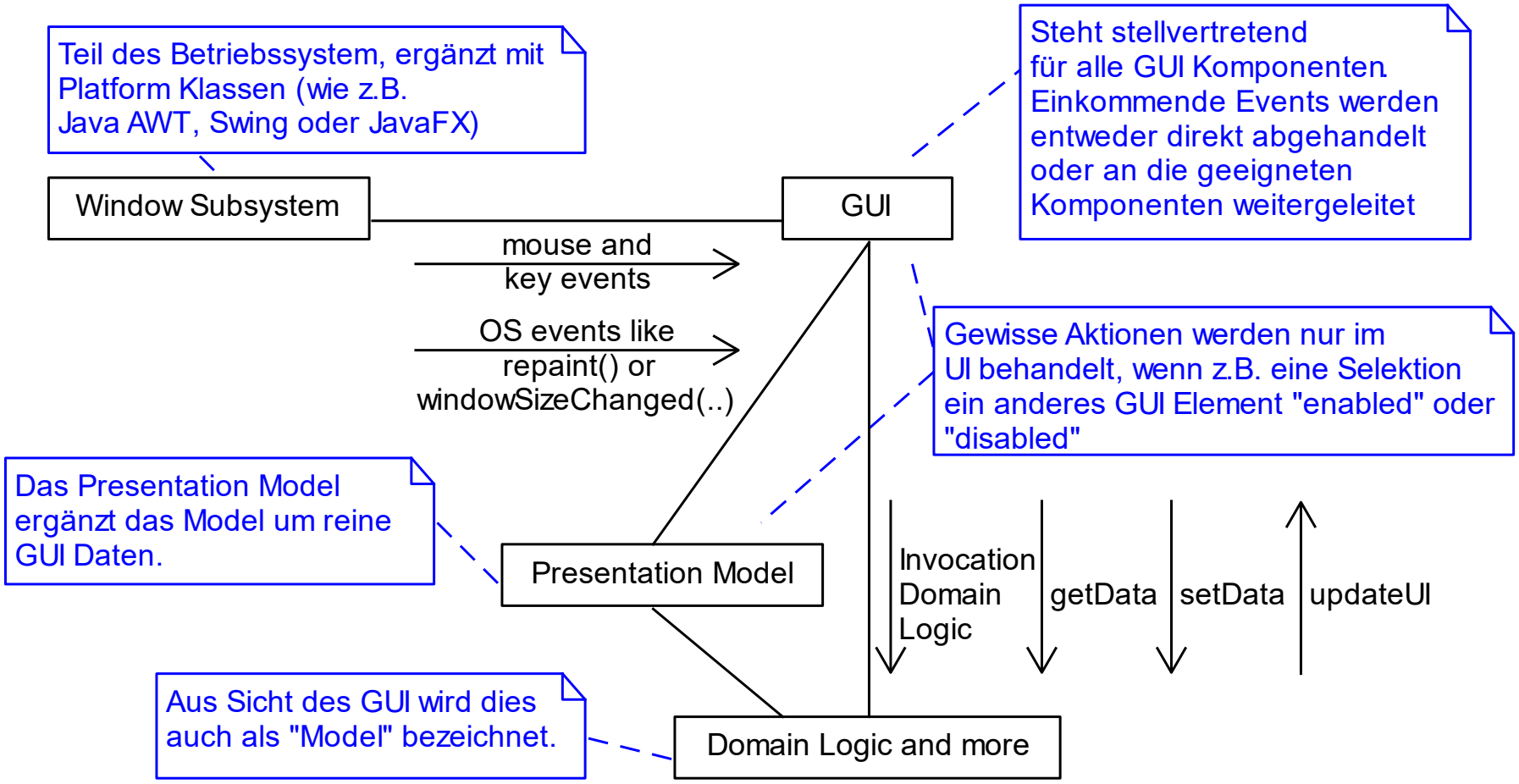
Domänen Modell GUI

- Das Domänenmodell einer GUI Anwendung enthält alle relevanten Konzepte
- Das GUI ist typischerweise aus Komponenten aufgebaut. Diese werden vom Operating System und von GUI Frameworks zur Verfügung gestellt, können aber auch selber entwickelt werden.



Zusammenarbeit GUI mit Betriebssystem und Domänenlogik

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie



Denkpause

Aufgabe 11.1 (5’)

Diskutieren Sie in Murmelgruppen folgende Frage:

- Wie wird die Schnittstelle zwischen Presentation bzw. UI und Domain Layer in Abhängigkeit der GUI-Varianten Desktop-App, Mobile-App und Web-App heute normalerweise umgesetzt?

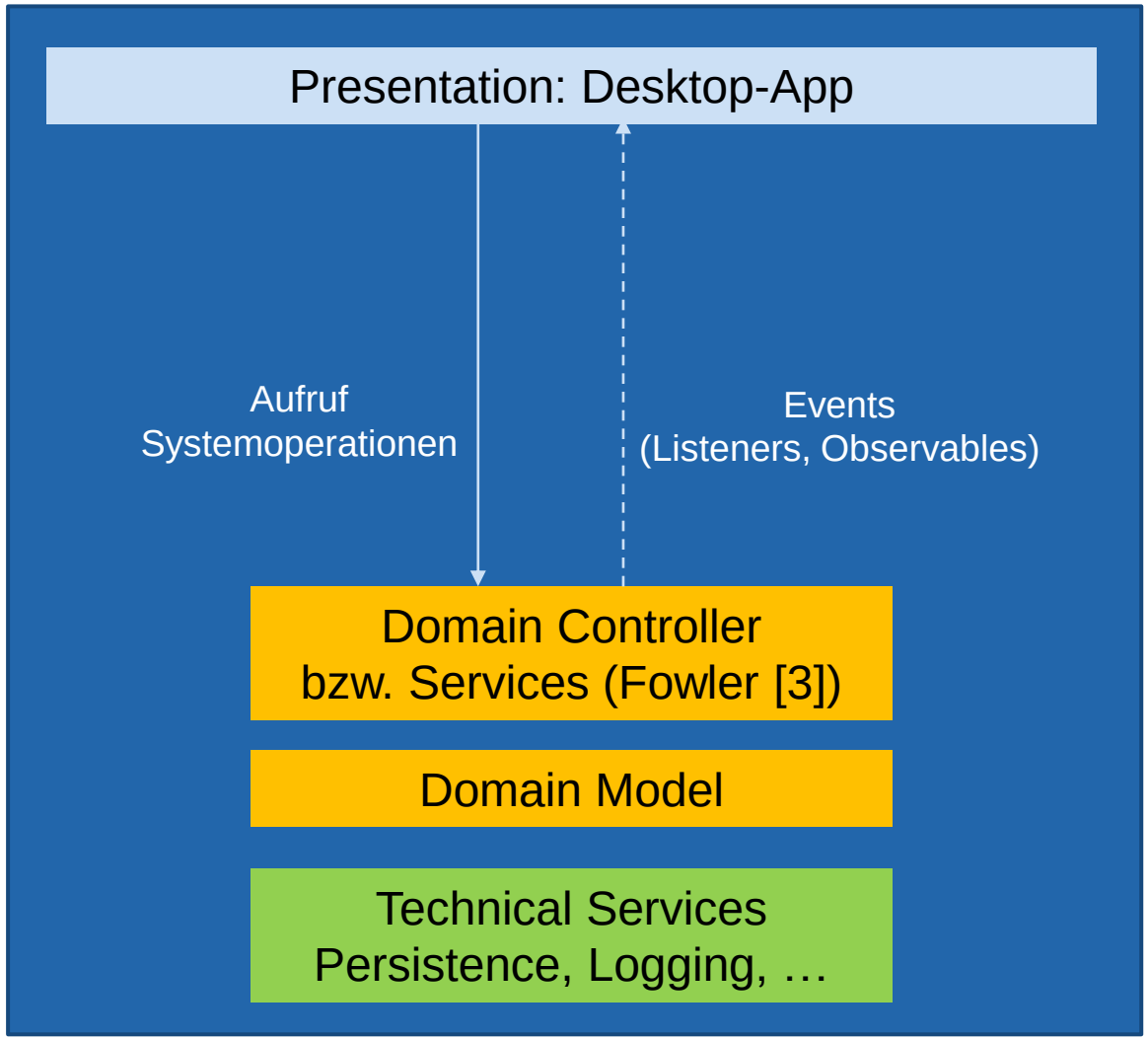
Denkpause

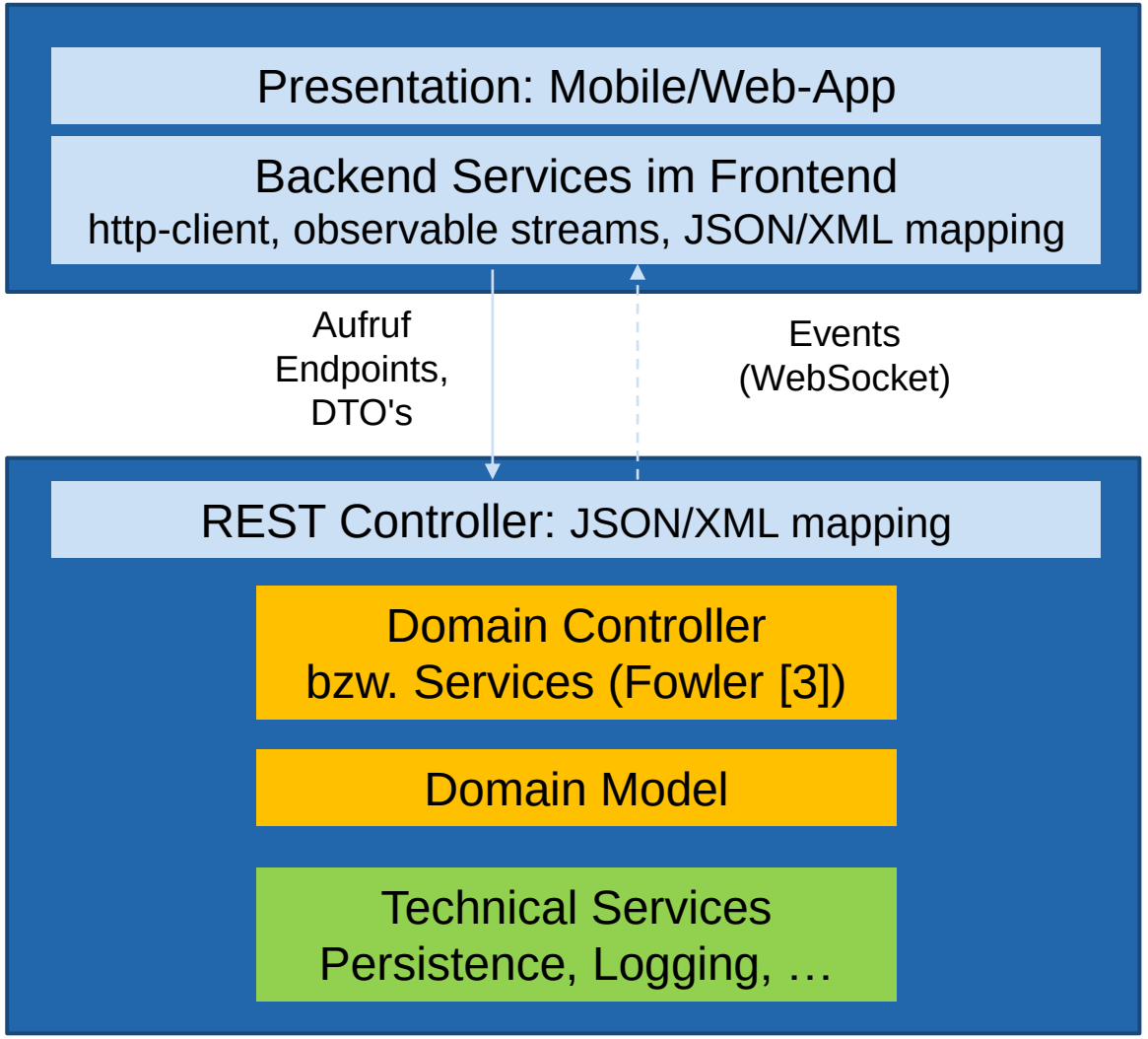
Aufgabe 11.1 – Musterlösung

Desktop-App: UI-Schicht ruft Methoden (Systemoperationen) im Larman Domain Controller auf (nach Fowler: Service). Die Domainschicht informiert die UI-Schicht über Events.

Mobile-App und Web-App: UI ruft Backend über publizierte Endpunkte auf (meistens REST), indem die zu übertragenden Daten in ein standardisiertes Format (z.B. JSON oder XML) als DTO's (Data Transfer Objects) gemappt werden. Die Domainschicht kann die UI-Schicht über WebSockets (Server-Push) informieren.

Schichtenarchitektur Vergleich





Agenda

1. Einführung in GUI-Architekturen
2. **Patterns in GUI-Architekturen**
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

Patterns für GUIs (1/2)

- Was und wie können Patterns verwendet werden, um GUIs zu strukturieren?
- Folgende Fragen sollen beantwortet werden:
  - Wer besitzt welche Teile der Daten?
  - Wer hat Zugriff auf die Daten? Wie?
  - Wer verarbeitet die Daten?
  - Wer präsentiert welche Teile der Daten?
  - Wer kann Daten verändern?
  - Wer wird informiert, nachdem Daten verändert wurden?

Patterns für GUIs (2/2)

- Observer
  - Wird über Änderungen an Objekten Informiert ohne observiertes Objekt im GUI zu instanziieren.
- Model View Controller (MVC) & Friends
  - Separierung von Zustand, Präsentation und Logik.
- Composite
  - Die gesamte View-Hierarchie wird in einer Composite-Struktur gerendert. Die View-Fläche muss unter den vorhanden Elementen aufgeteilt werden (z.B. über *Responsive Layout Manager*).

Observer Pattern in Java : Property Change Listener


```
public class StudentModel {
    private String name;

    private final PropertyChangeSupport changes = new PropertyChangeSupport( this );

    public void setName(String name) {
        String oldName = this.name;
        this.name = name;
        changes.firePropertyChange( "name", oldName, name );
    }

    public void addPropertyChangeListener( PropertyChangeListener listener ) {
        changes.addPropertyChangeListener( listener );
    }
}
```

```
    studentModel.addPropertyChangeListener( e -> {
        studentViewModel.onStudentNameChanged(e.getNewValue().toString());
    });
```

Weitere Anwendungen von Observer Pattern

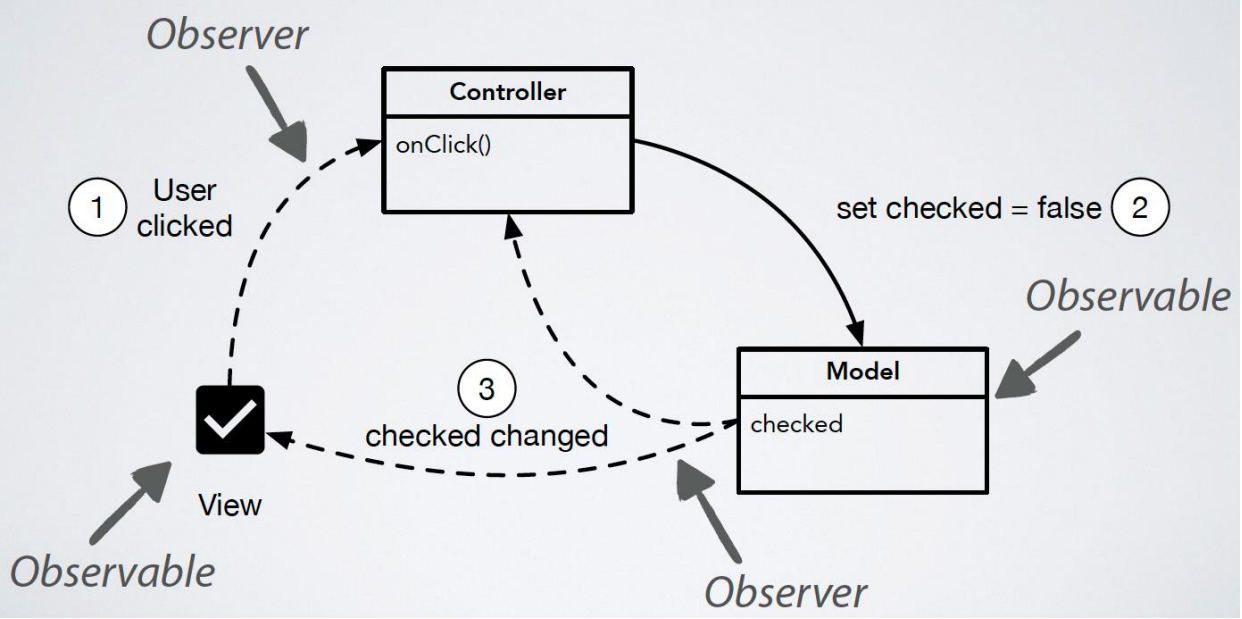
- Event Handling
  - Jeder onClickListener() etc. ist eine Anwendung des Observer Patterns.
- RxJava
  - Die Datenquellen werden aus diesem Grund Observables benannt.
- Event Bus
  - Mehrere Observables publizieren Events auf den gleichen Kanal.
  - Mehrere Observer werden mit dem Kanal verbunden und reagieren auf die für sie relevanten Events.
- Data Binding
  - JavaFX Properties, Java Beans Property Change Listener.

Agenda

1. Einführung in GUI-Architekturen
2. Observer Pattern in GUI-Architekturen
3. **Architektur-Patterns MVC, MVP und MVVM**
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

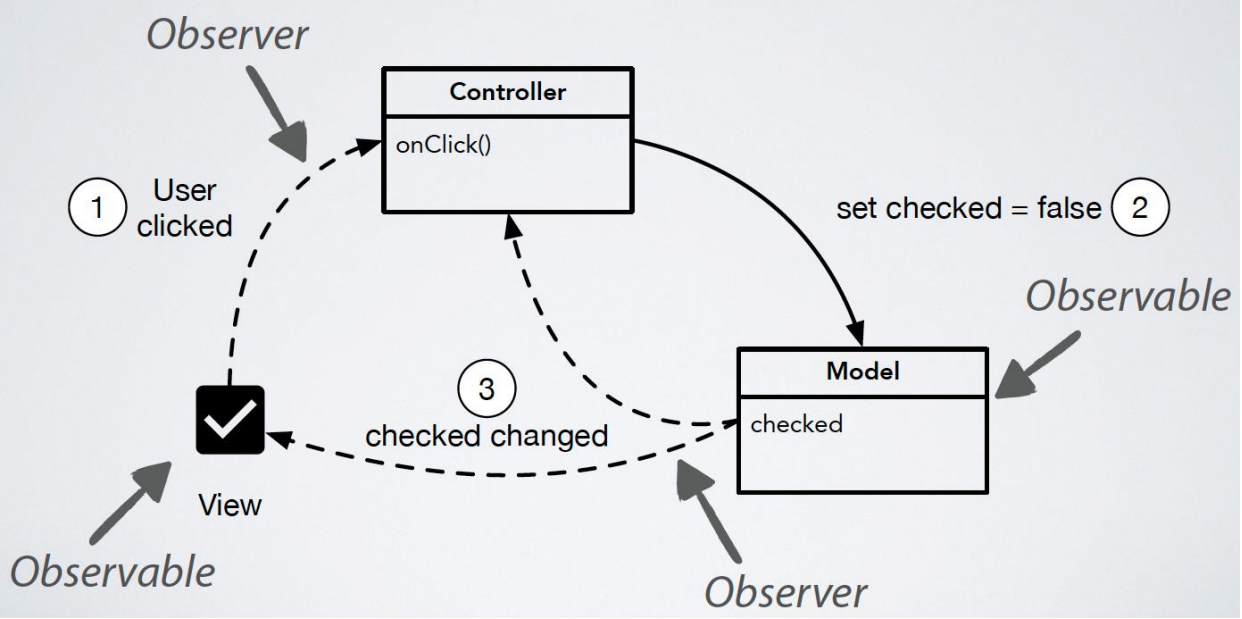
Model View Controller (MVC) (1/2)

- Trennung von Zustand, Präsentation und Logik
  - Bessere Testbarkeit
  - Viele unterschiedliche Interpretationen
- View: Was der Benutzer sieht
  - z.B. Checkbox
- Model: Anzuzeigende Daten
  - checked = true/false



Model View Controller (MVC) (2/2)

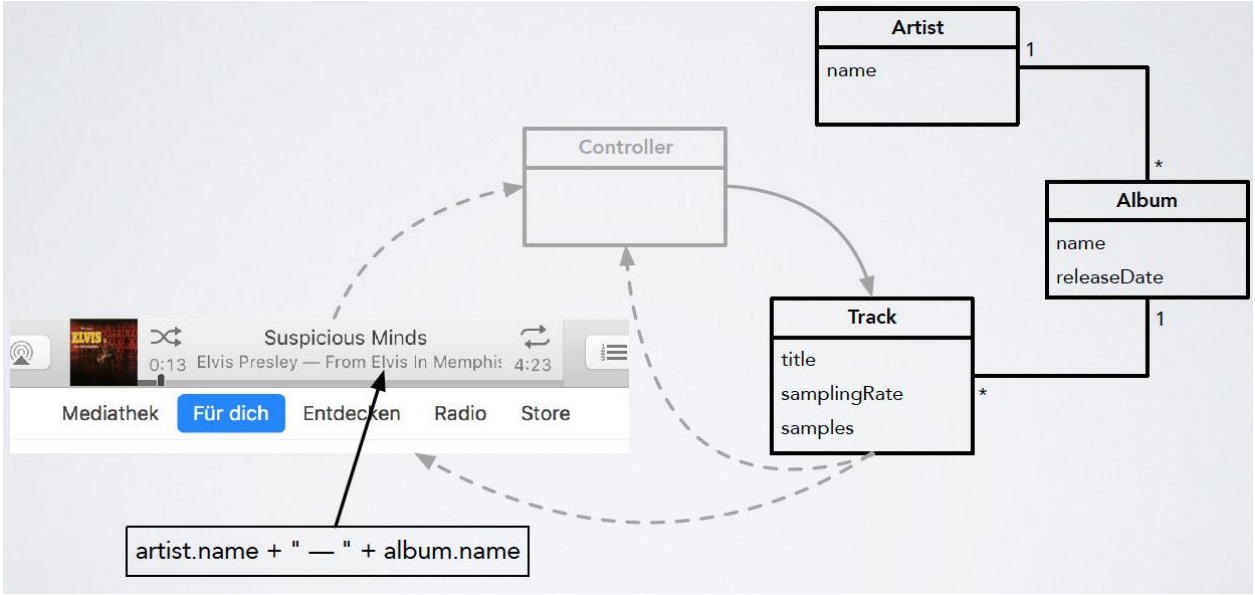
- Controller
  - Behandelt Benutzer-Ereignisse
  - Koordiniert Aktionen mit anderen Controllern in der Anwendung
  - Enthält die Logik für die Veränderung des Models (über Domain Controller)



- Observer-Pattern ist integraler Teil von MVC
  - Das Model (Observable) informiert die View (Observer) über Änderungen
  - Der Controller sollte nicht direkt mit der View kommunizieren, sondern indirekt über die Aktualisierung des Model.

Das Modell

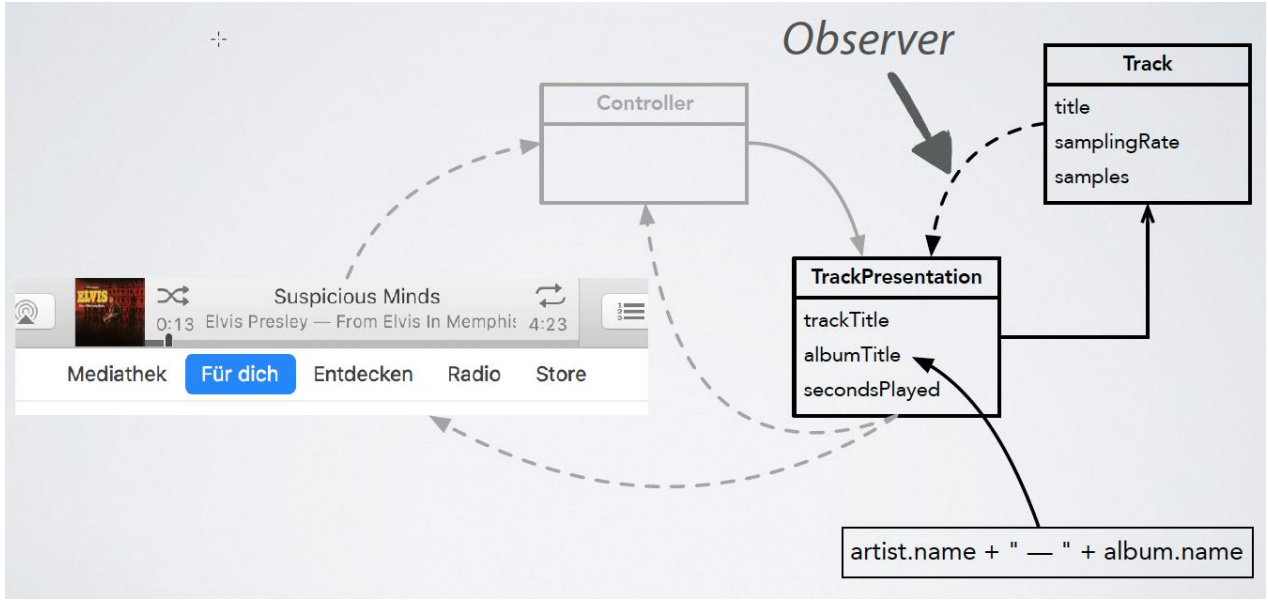
- Model -> Domain Model
  - Ein Model pro Anwendung
- Wird schnell unübersichtlich
  - Wo sollen berechnete Werte gespeichert werden?
  - Wo der Zustand des UI's?
  - Wo I18n (Internationalisierung)?



- Diese UI-Daten ruinieren das Domain Model.
  - Das Domainmodel soll nur fachliche Daten enthalten.

Das Presentation Model

- Das Presentation Model löst das Problem
  - Wird vor dem Domain Model positioniert
- Enthält alle darzustellenden Daten plus den Zustand des UI's
  - Minimiert die Logik des Views
  - Hält das Domain Model sauber

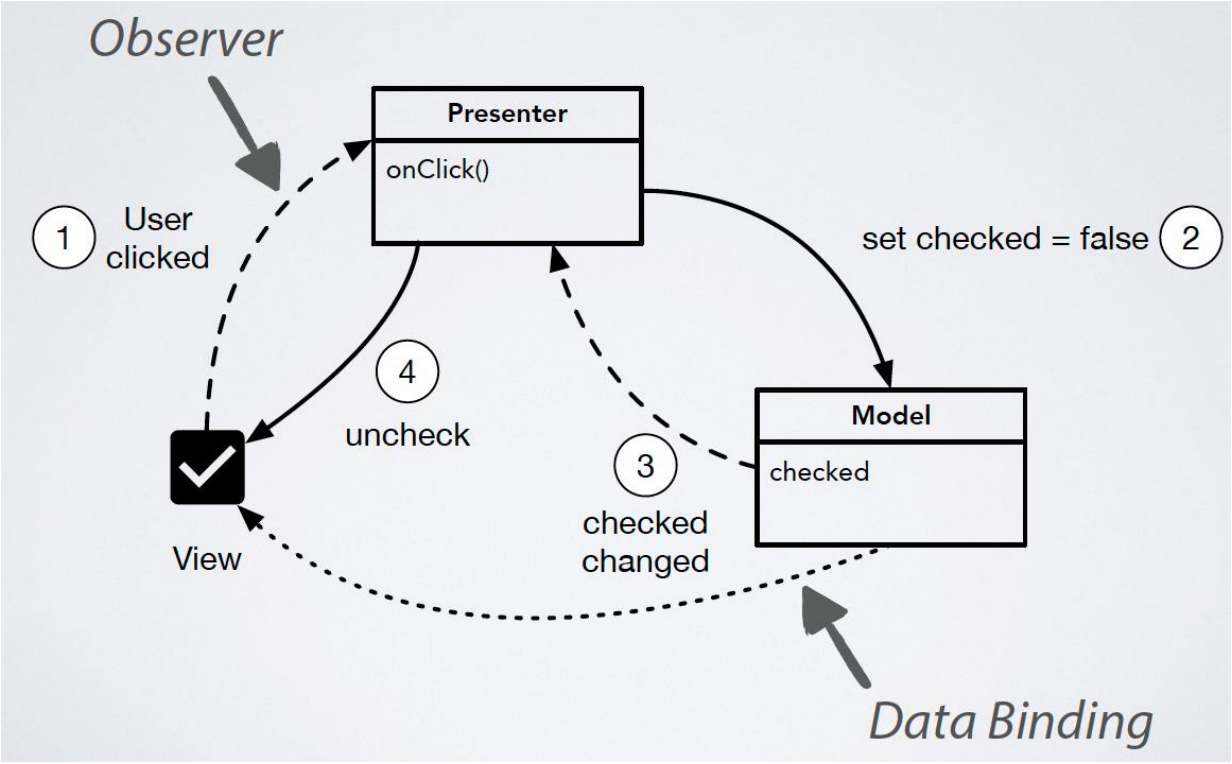


Model View Presenter (MVP)

- MVP ist die ursprüngliche Bezeichnung
- Zwei Varianten
  - Supervising Controller
  - Passive View

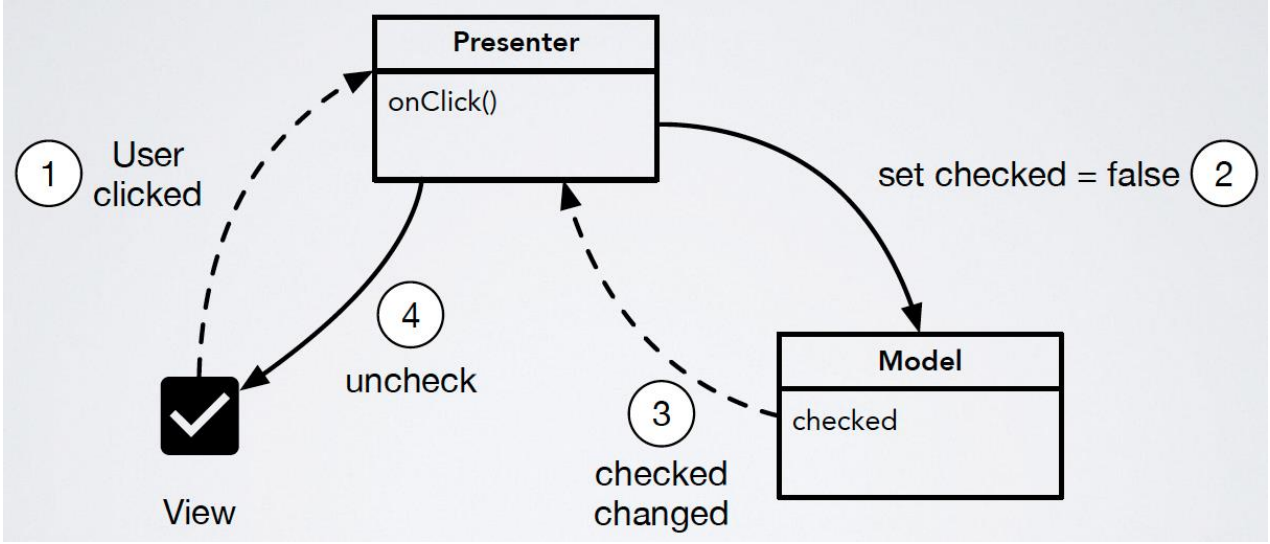
MVP - Supervising Controller

- Unterschied MVC zu Supervising Controller
  - View wird aktualisiert mittels Data Binding anstelle von Observer
  - Presenter manipuliert die View direkt
- Verwendung eines Presentation Models ist empfohlen.
- Erhöht die Testbarkeit, weil sich die Logik nicht mehr in der View befindet.



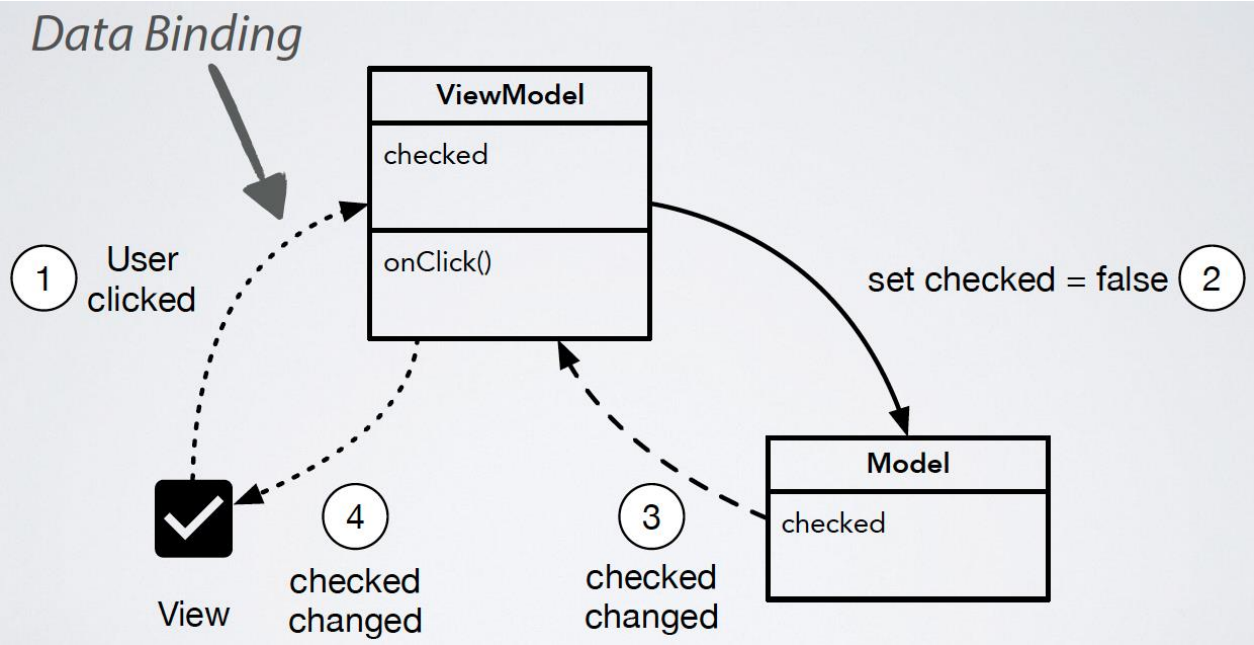
MVP - Passive View

- Im Vergleich zu Supervising Controller
  - Die View enthält keine Logik
  - Kann sich selber nicht aktualisieren
  - Der Presenter enthält alle Logik für die Aktualisierung des Views
- Verwendung eines Presentation Models ist empfohlen
- Erhöht die Testbarkeit, weil sich die Präsentations-Logik nicht mehr in der View befindet



Model View ViewModel (MVVM)

- Eng verwandt mit dem Supervising Controller
- ViewModel
  - Ersetzt den Supervising Controller
  - Enthält ein Presentation Model
  - View kommuniziert mit dem ViewModel durch DataBinding




```
private void initBindings() {
    label.textProperty().bind(studentViewModel.labelTextProperty());
    input.textProperty().bindBidirectional(studentViewModel.inputTextProperty());
    button.disableProperty().bind(studentViewModel.buttonDisabledProperty());
}
```

Agenda

1. Einführung in GUI-Architekturen
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. **Kurzer Recap JavaFX**
5. Wrap-up und Ausblick

JavaFX und JDK

- JavaFX 8 ist/war ein Teil von Oracle JDK 8, 9 und 10
- JavaFX-Library muss separat ins Projekt eingebunden werden.
  - Ab Open JDK 8
  - Ab Oracle JDK11
- JavaFX Download (OpenFX) von
  - https://gluonhq.com/products/javafx/
  - JavaFX 11 LTS -> Long Term Support

JavaFX kann **direkt** gestartet werden

JavaFX benötigt einen **Launcher** bei non-modularen Projekten

Oracle JDK8 -10

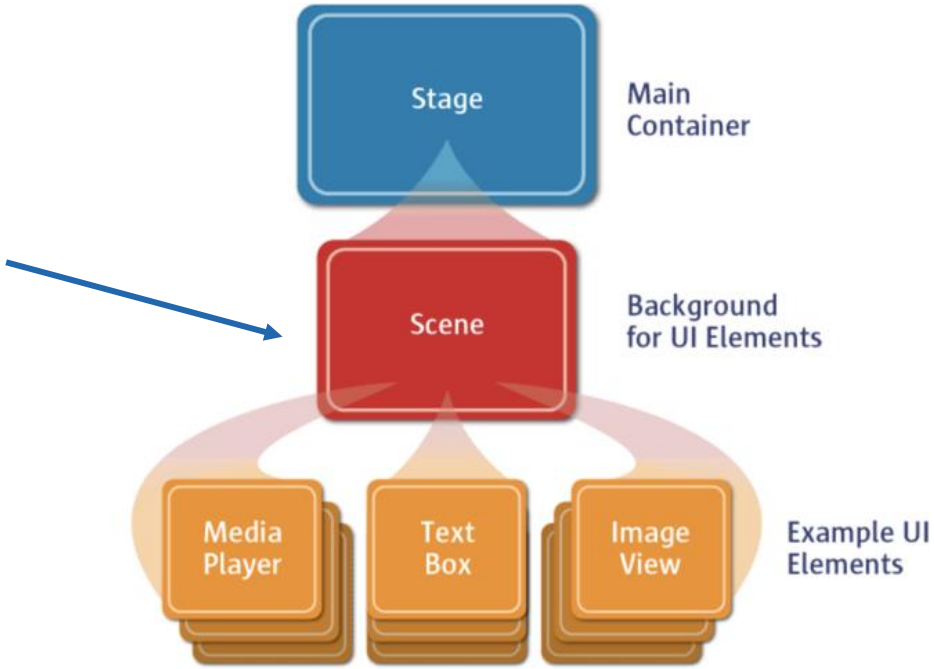
OpenJDK 8--xx
und OracleJDK
11 – xx

JavaFX 8
Teil des JDK8-10

JavaFX Version xx
JavaFX Libraries
müssen eingebunden
werden

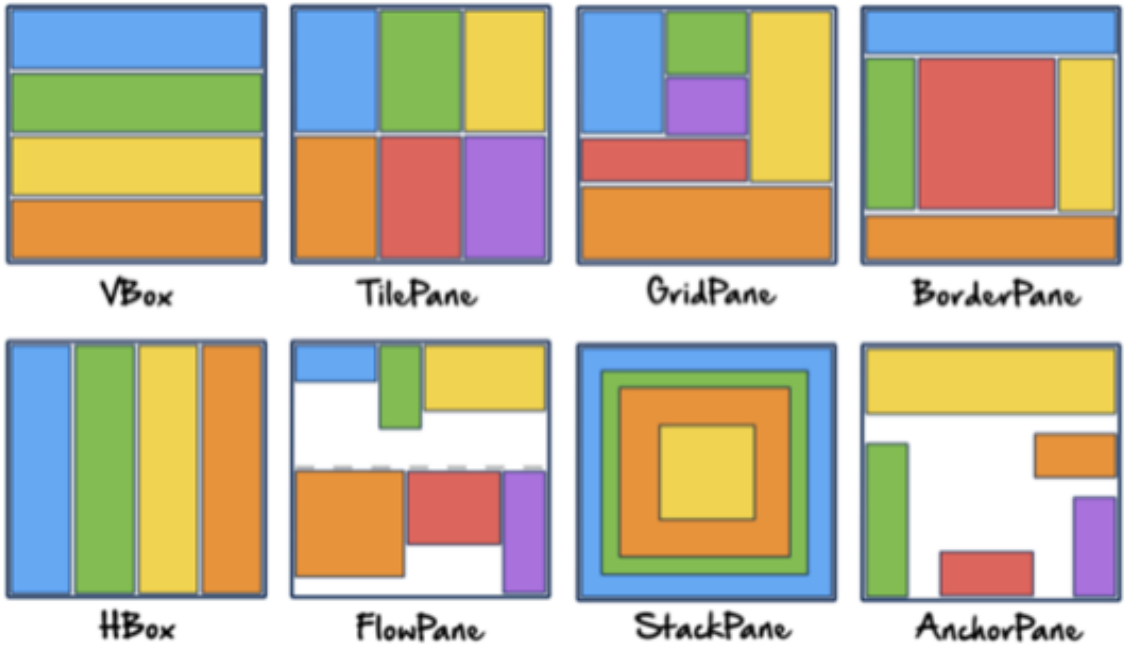
Hierarchischer Aufbau

- Stage (Fenster)
  - Ein Fenster in JavaFX besteht aus Objekten, die ineinander verschachtelt sind.
  - Auf der obersten Ebene steht ein Objekt vom Typ Stage (engl. für Bühne), welches das Fenster als Ganzes repräsentiert.
  - Man nennt das auch ein Top-Level-Fenster.
- Scene (Fenster Inhalt)
  - Die zentrale Container-Komponenten auf der alle anderen Nodes platziert werden.
- Nodes
  - Die Nodes bilden den Scene-Graph und getScene() liefert die Scene.



Layoutmanager

- Innerhalb des Scene-Objekts wird so etwas wie ein Setzkasten aus verschiedenen strukturellen Objekten hergestellt.
- Layout-Klassen von JavaFX
  - Man nennt sie auch Panes (engl. für Fensterleiste)
  - Unterklassen von Java-Klasse: javafx.scene.layout.Pane



Styling mit CSS-Stylesheets

- Trennung von Layout und Stil.
  - Layout meint hier die Wahl und Anordnung der Elemente (z.B. alle Buttons unten ausgerichtet).
  - Stil bezieht sich auf Farbe, Form, Schrift etc.
- Regeln
  - Ein Stylesheet ist eine einfache Text-Datei mit Endung .css
  - Der Selektor besagt, für welche Art Elemente diese Regel *feuert*.
  - Mit den Eigenschaft-Wert-Paaren setzt man dann die Eigenschaften.
  - Jede Regel hat das Format:

```
SELEKTOR {
EIGENSCHAFT: WERT;
EIGENSCHAFT: WERT;
...
}
```

```
.button {
-fx-font-size: 14px;
}
```

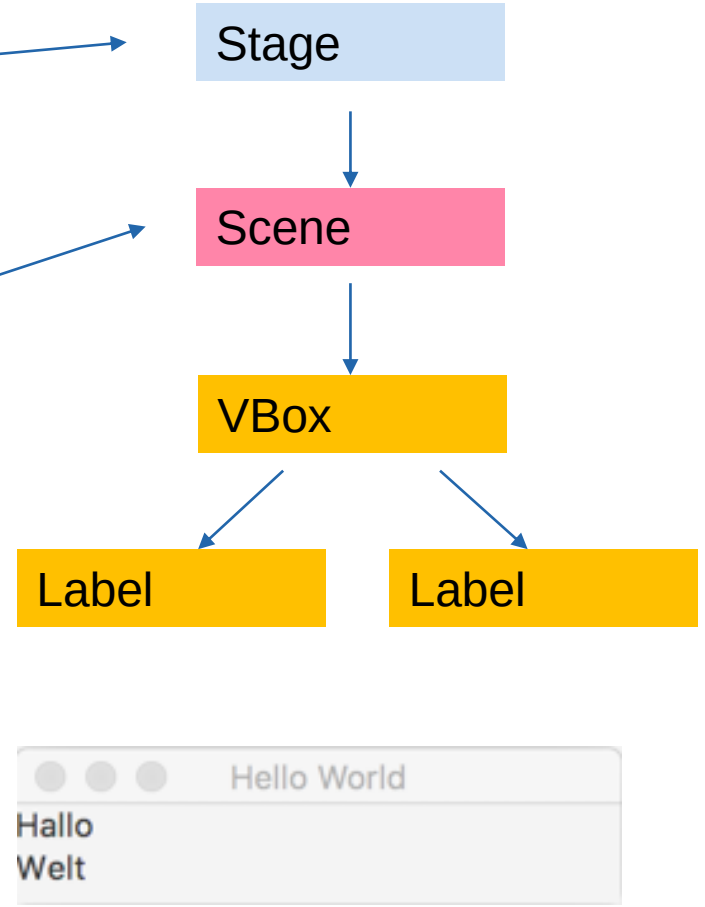
JavaFX Hello World


```
public class HelloWorld extends Application {

  public void start(Stage primaryStage) {
    Label l1 = new Label("Hallo");
    Label l2 = new Label("Welt");
    VBox root = new VBox();
    root.getChildren().add(l1);
    root.getChildren().add(l2);

    Scene scene = new Scene(root, 240, 40);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Hello World");
    primaryStage.show();
  }

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

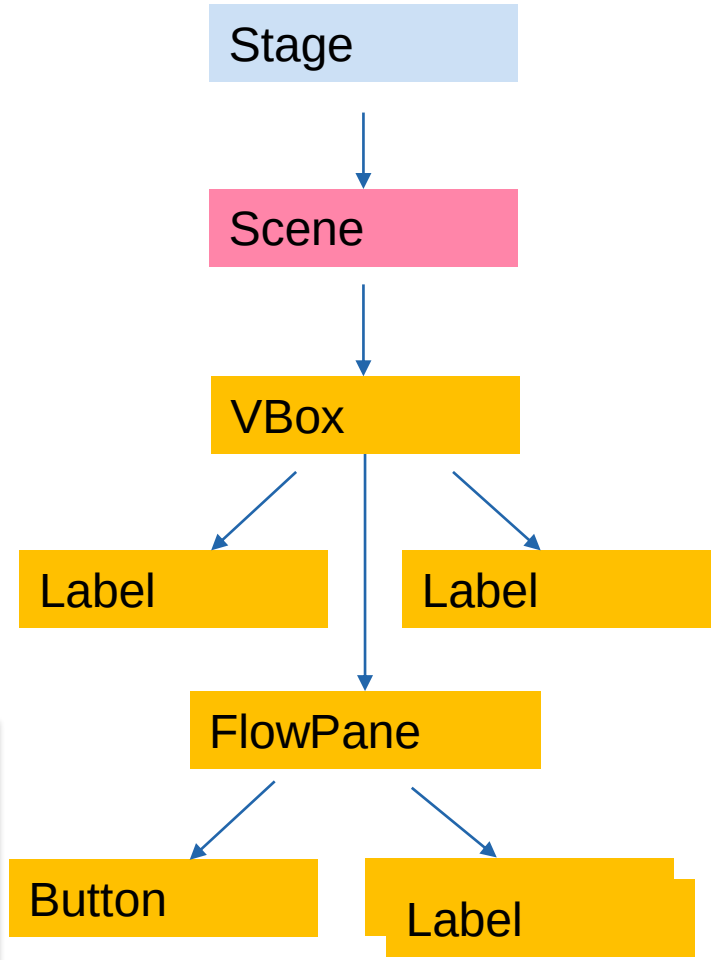


Ereignisorientierte Programmierung - Action Handling (1/2) Button mit setOnAction()

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Anonyme innere Klasse
  - mit EventHandler<ActionEvent>

```
Pane createButton() {
  final Button btn = new Button();
  btn.setText("Add 'Hello World' Label");
  final FlowPane pane = new FlowPane();
  pane.setPadding(new Insets(7, 7, 7, 7));
  pane.getChildren().add(btn);
  // ActionHandler registrieren
  btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent even
      pane.getChildren().add(
          new Label("- Hello World! -"))
    }
  });
  return pane;
}
```



Ereignisorientierte Programmierung - Action Handling (2) Button mit setOnAction () Lambda

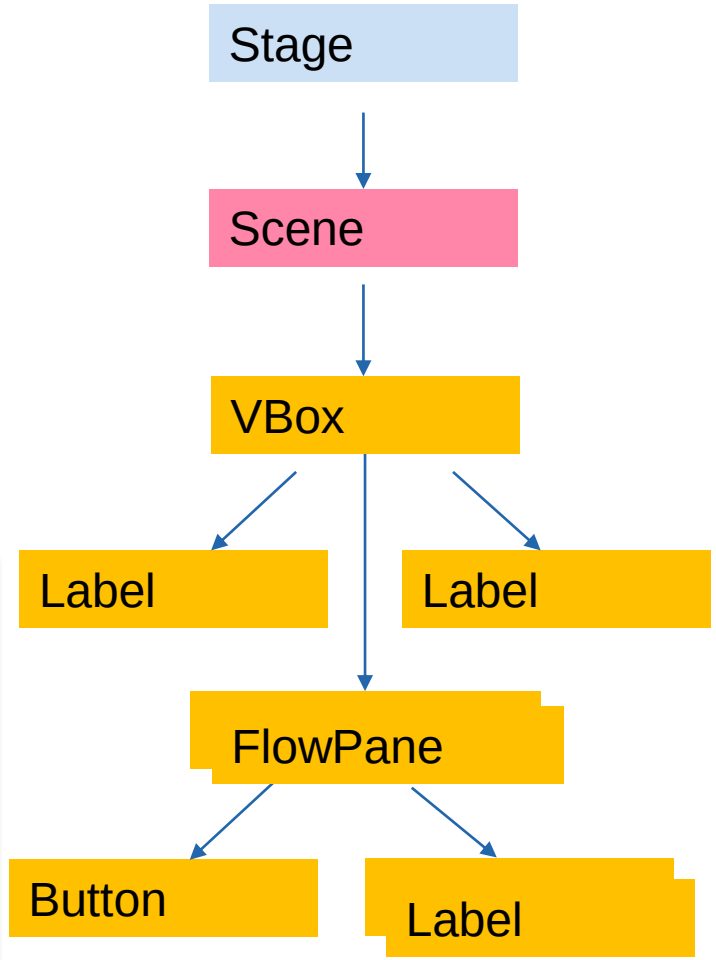
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Einfacher mit Lambda

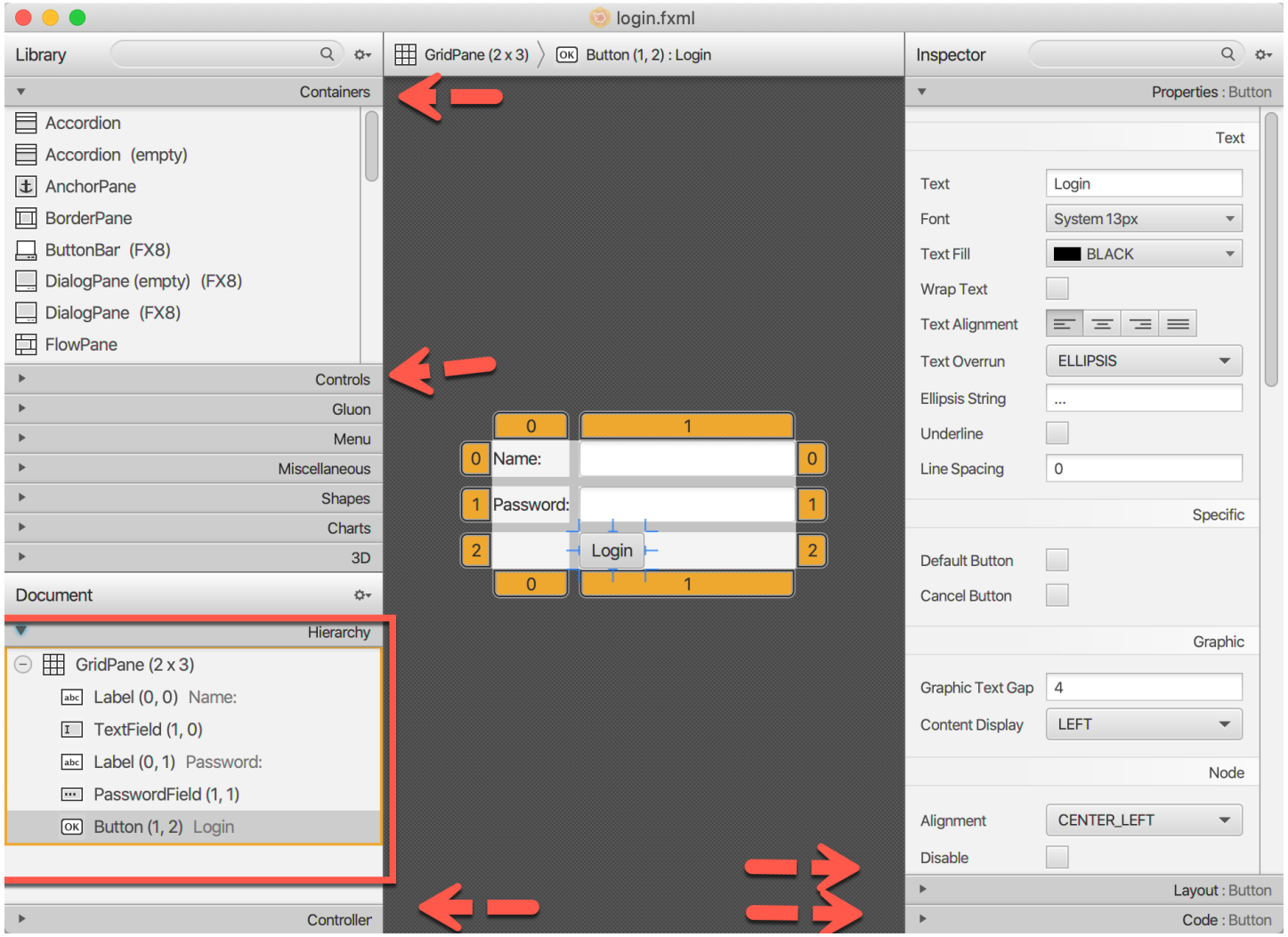
```
Pane createButtonLambda() {
  final Button btn = new Button();
  btn.setText("Add 'Hello World' Label");
  final FlowPane pane = new FlowPane();
  pane.setPadding(new Insets(7, 7, 7, 7));
  pane.getChildren().add(btn);

  // ActionHandler registrieren
  btn.setOnAction(event -> pane.getChildren()
    .add(new Label("- Hello World! -"))
  );

  return pane;
}
```



Scene Builder als GUI-Design-Tool



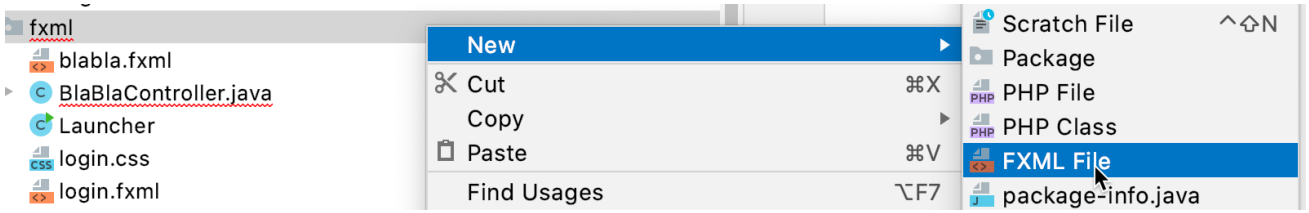
https://gluonhq.com/products/scene-builder/

Vorgehen für die Erstellung eines Login-Dialogs (1/7)

1. Leere Controller Klasse erstellen

   Beispiel: LoginController

```
public class LoginController {
}
```



2. fxml-Datei login.fxml erstellen

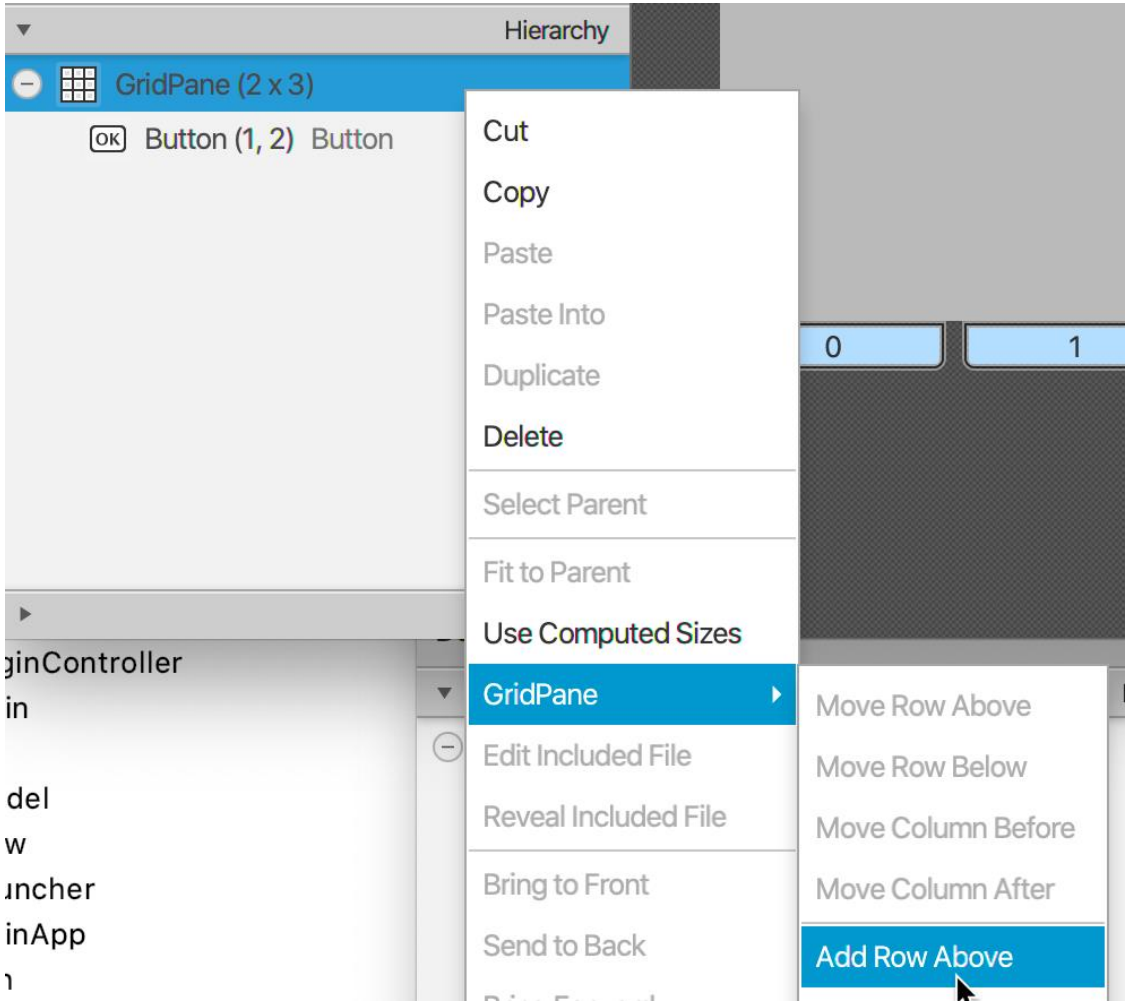
   – Beispiel: **login.fxml**
   – AnchorPane durch GridPane ersetzen

```
<AnchorPane xmlns="http://javafx.com/javafx"
            xmlns:fx="http://javafx.com/fxml"
            fx:controller="sample.fxml.Login"
            prefHeight="400.0" prefWidth="600.0">

</AnchorPane>
```

Vorgehen für die Erstellung eines Login-Dialogs (2/7)

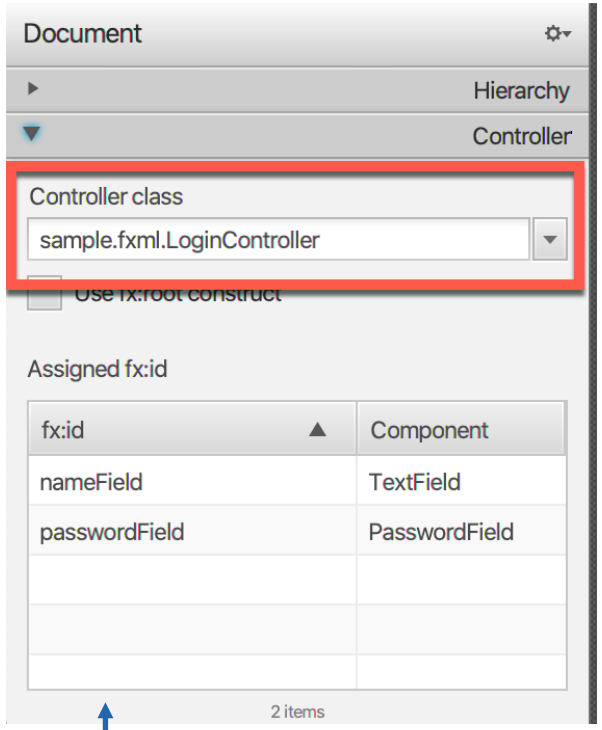
- Im Scene Builder Add Row und Add Column verwenden
  - Ein Grid 2 x 3 erstellen
  - Anschliessend Label, Textfield und Button positionieren



Vorgehen für die Erstellung eines Login-Dialogs (3/7) Bindings einstellen

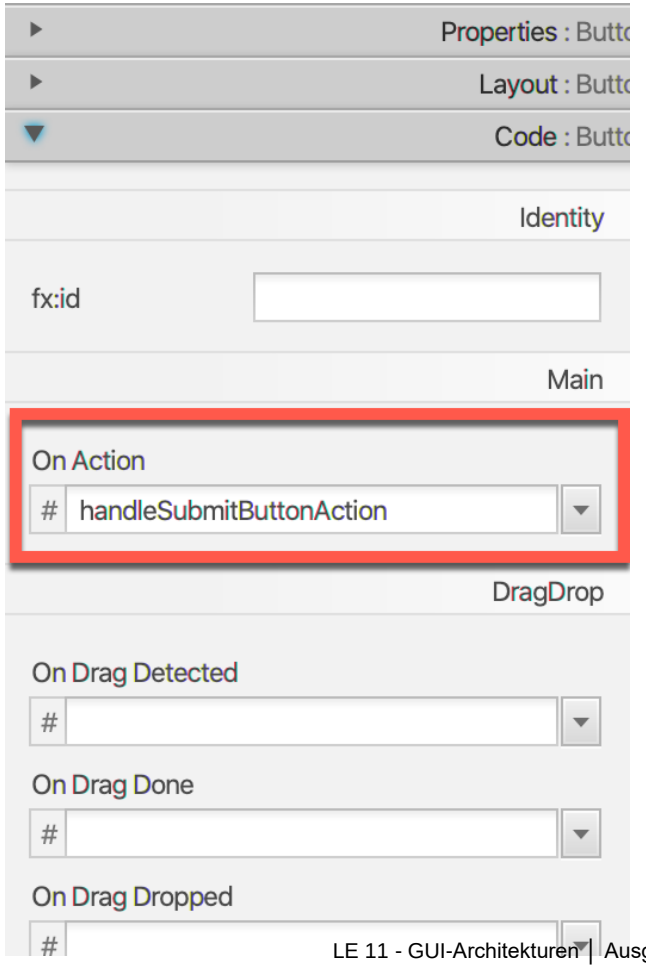
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Java Controller Klasse

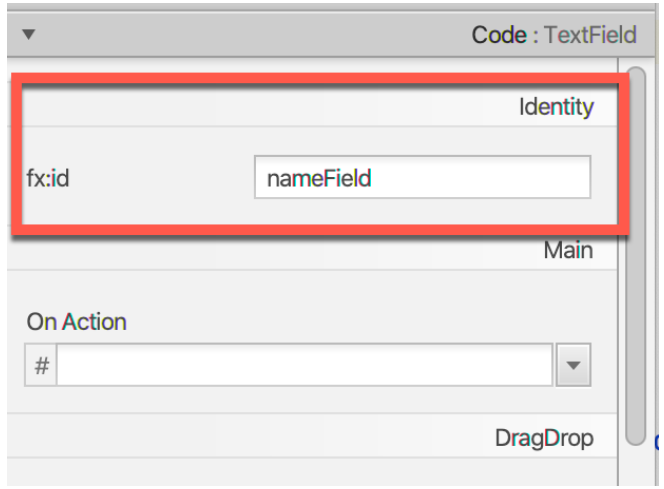


Gebundene Felder

Methoden Name bei Action



Id TextField

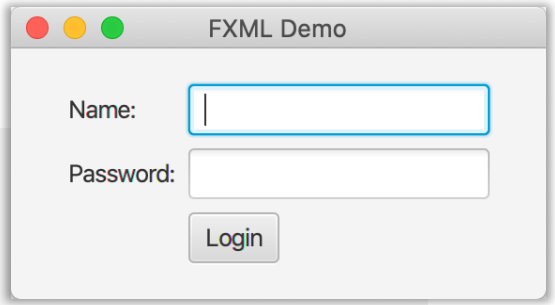


nameField
passwordField

Vorgehen für die Erstellung eines Login-Dialogs (4/7) FXML und Darstellung

- Scene Builder erstellt eine fxml-Datei

Package Controller




```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<GridPane alignment="CENTER" hgap="7.0" vgap="7.0" xmlns:fx="http://javafx.com/fxml/1"
      xmlns="http://javafx.com/javafx/2.2" fx:controller="sample.fxml.LoginController">
  <children>
     <Label text="Name:" GridPane.columnIndex="0" GridPane.rowIndex="0" />
     <TextField fx:id="nameField" GridPane.columnIndex="1" GridPane.rowIndex="0" />
     <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="1" />
     <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="1" />

     <Button onAction="#handleSubmitButtonAction" text="Login" GridPane.columnIndex="1"
             GridPane.rowIndex="2" />
  </children>
</GridPane>
```

Vorgehen für die Erstellung eines Login-Dialogs (5/7)
FXML **Supervising**-Controller Klasse

- Die Annotation `@FXML` verbindet den Java Code mit der fxml-Datei

```
public class LoginController {
    @FXML
    private PasswordField passwordField;

    @FXML
    private TextField     nameField;

    @FXML
    protected void handleSubmitButtonAction(ActionEvent event)
    {
        System.out.println("Signin button user input: "
                + nameField.getText() + " pwd: "
                + passwordField.getText());
    }
}
```

Vorgehen für die Erstellung eines Login-Dialogs (6/7) Laden der FXML Datei

- Die fxml-Datei befindet sich in im gleichen Ordner
- Bei Maven-Projekten ist die fxml-Datei im `Resource` Ordner.
  - Ordner entsprechend anpassen, gleicher Ordner wie Controller

```
public void start(Stage stage) throws Exception {

    final Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));
    stage.setScene(new Scene(root, 460, 180));
      // Load CSS
      // stage.getScene().getStylesheets().add(getClass().getResource("login.css").toExternalForm());

    stage.setTitle("FXML Demo");
    stage.show();
}
```

Vorgehen für die Erstellung eines Login-Dialogs (7/7) Laden der FXML Datei und Zugriff auf Controller

- Für weitere Initialisierung wird manchmal der Zugriff auf den Supervising-Controller benötigt.

```
//create a root and load fxml code
FXMLLoader loader = new FXMLLoader(getClass().getResource("login.fxml"));
final Parent root = loader.load();

// get the LoginController
LoginController loginController = (LoginController) loader.getController();
```

Unidirektionales Binding

- prop2 ist gekoppelt an prop1

```
public class UnidirectionalBinding {
  public static void main(String[] args) {
    SimpleIntegerProperty prop1 = new SimpleIntegerProperty();
    SimpleIntegerProperty prop2 = new SimpleIntegerProperty();
    prop2.bind(prop1); // prop2 ist an prop1 gekoppelt (nicht umgekehrt)
    System.out.println(prop1.get() + " / " + prop2.get()); // => 0 / 0
    prop1.set(101);
    System.out.println(prop1.get() + " / " + prop2.get()); // => 101 / 101
    try {
      prop2.set(49);
    } catch (Exception e) {
      System.out.println("das funktioniert also nicht");
    }
    prop2.unbind();
    prop2.set(73);
    System.out.println(prop1.get() + " / " + prop2.get()); // => 101 / 73
  }
}
```

Bidirektionales Binding

- Properties können sich gegenseitig aktualisieren

```
public class BidirectionalBinding {
  public static void main(String[] args) {
    SimpleIntegerProperty prop1 = new SimpleIntegerProperty();
    SimpleIntegerProperty prop2 = new SimpleIntegerProperty();
    prop1.bindBidirectional(prop2); // oder prop2.bindBidirectional(prop1);
    System.out.println(prop1.get() + " / " + prop2.get()); // => 0 / 0
    prop1.set(101);
    System.out.println(prop1.get() + " / " + prop2.get()); // => 101 / 101
    prop2.set(49);
    System.out.println(prop1.get() + " / " + prop2.get()); // => 49 / 49
  }
}
```

Observable Collections

- Containerklassen mit Benachrichtigungsfunktionalität
  - `ObservableList`, `ObservableSet` und `ObservableMap`
  - Konstruktion durch Erzeugungsmethoden aus FXCollections

```
final String[] content = { "Orig1", "Orig2" }
final ObservableList<String> observableList =
                            FXCollections.observableArrayList(content);

observableList.addListener((ListChangeListener<String>) change ->
                System.out.println("Changed to: " + change.getList()));

observableList.addAll("A", "B", "C");
observableList.removeAll("Orig1", "Orig2");
observableList.add("1");
observableList.add("2");
```

```
Changed to: [Orig1, Orig2, A, B, C]
Changed to: [A, B, C]
Changed to: [A, B, C, 1]
Changed to: [A, B, C, 1, 2]
```

Agenda

1. Einführung in GUI-Architekturen
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
5. **Wrap-up und Ausblick**

Wrap-up

- GUI-Architekturen werden in der Presentation-Schicht bzw. UI angewendet.
- Die gängigen GUI-Architekturen basieren auf bewährten Patterns.
- Die Patterns erlauben die Umsetzung von «Separation of Concern». Das bekannteste Pattern ist das MVC. Weitere Pattern sind das MVP und MVVM.
- Die Reaktion auf Ereignisse kann durch das Observer-Pattern hergestellt werden. Das Pattern wird in vielen Frameworks angewendet.
- Bei der Frontend-Programmierung steht die Reaktion auf Ereignisse im Vordergrund. Die Umsetzung kann mit Hilfe von anonymen Klassen oder vereinfacht durch Lambda-Ausdrücke erreicht werden.
- Bei JavaFX kann ein View imperativ durch die Instanziierung der Steuerelemente erstellt werden. Mit Hilfe von FXML steht eine deklarative Möglichkeit zur Verfügung.

Ausblick

- In der nächsten Lerneinheit werden wir:
  - das Thema der Persistenz vertiefen.

Quellenverzeichnis

[1] Martin Fowler, GUI Architectures, https://martinfowler.com/eaaDev/uiArchs.html, 2006

[2] Gluon, JavaFX, https://gluonhq.com/products/javafx/

[3] Fowler, M.: Patterns of Enterprise Application Architecture, Addison Wesley, 1. Auflage, 2002