

Grafische Benutzeroberflächen mit Java FX

Programmiermethodik 2

Zum Nachlesen:

Carl Dea et al.: JavaFX 8: Introduction by Example, Apress, 2014
Oracle Docu: <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

Wiederholung

- Einführung
- Strategie
- Beobachter
- Model-View-Controller

Ausblick



Agenda

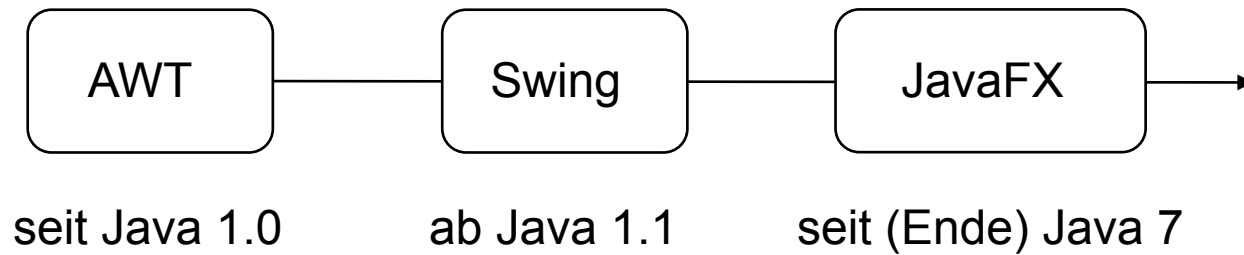
- Grafische Benutzeroberflächen mit JavaFX
- Fenster mit Knopf
- Anordnen von Komponenten
- Properties
- Tabellen
- Worker Threads
- Scene Builder



JavaFX

JavaFX

- Historische Entwicklung



JavaFX

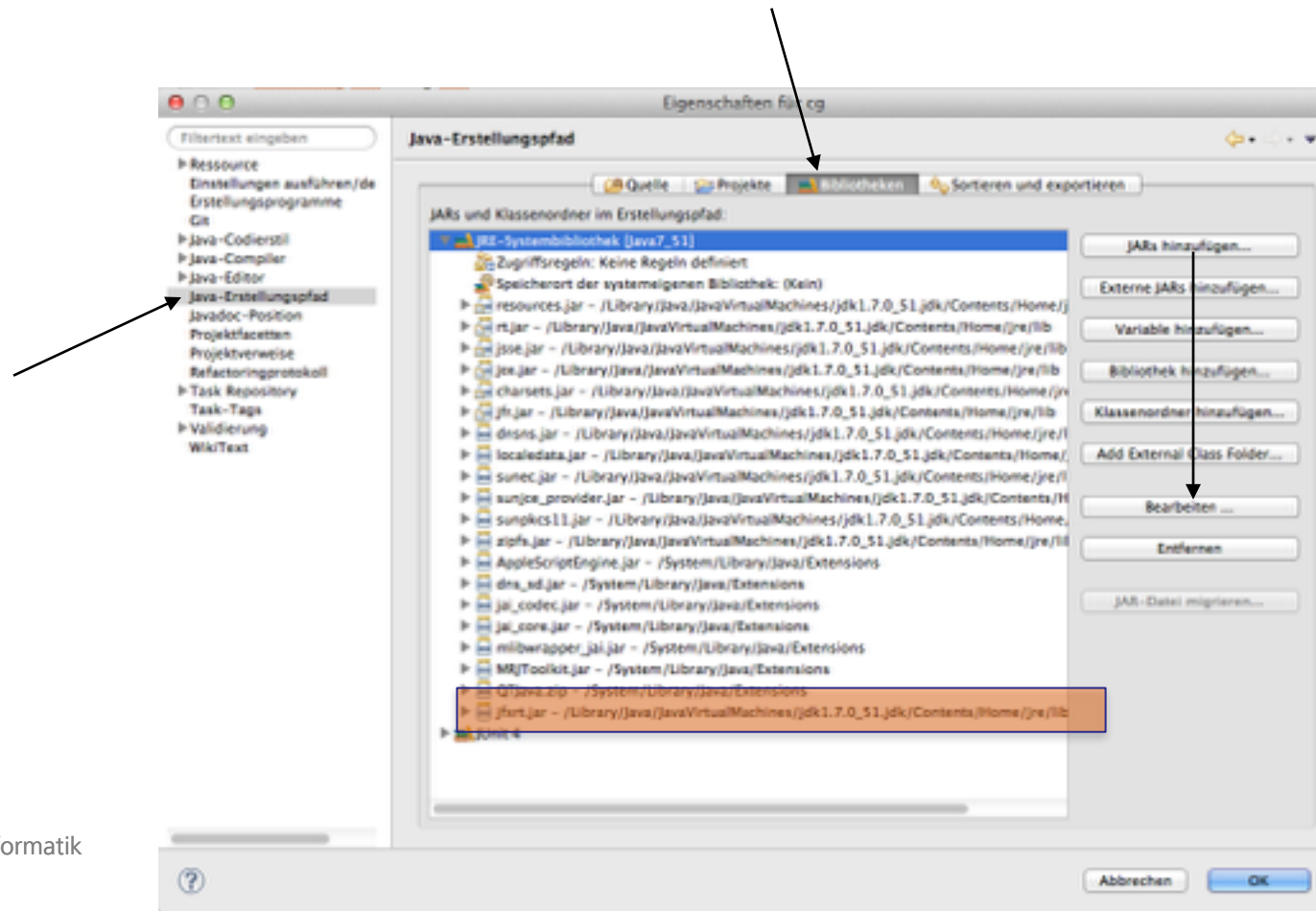
- ein Schweizer Taschenmesser



Quelle: Carl Dea: JavaFX 2.0 Introduction by Example, 2011, APress

Installation

- erst seit Java 8 als Standard dabei
- manchmal Probleme, das notwendige Jar zu finden
 - *jfxrt.jar*

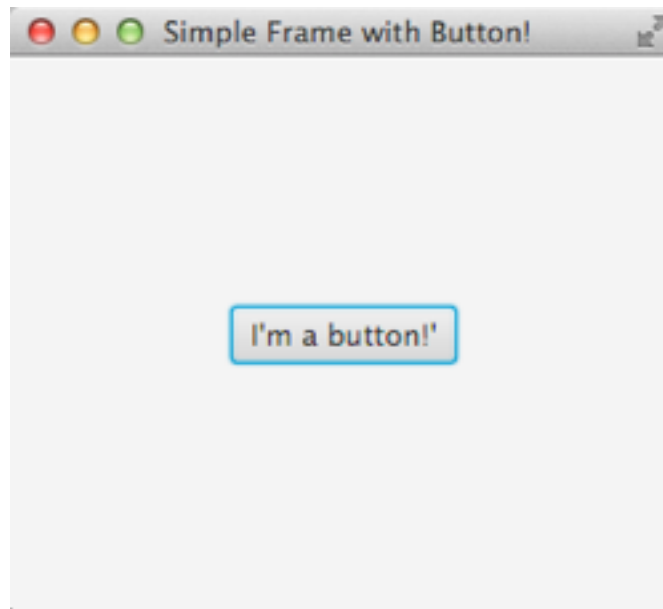




Fenster mit Knopf

Fenster mit Knopf

- Es soll ein Fenster mit einem Knopf erscheinen.



Fenster mit Knopf

```
public class FensterMitKnopf extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Fenster mit Knopf!");  
  
        ToggleButton knopf = new ToggleButton();  
        knopf.setText("Ich bin ein Knopf!");  
        StackPane wurzel = new StackPane();  
        wurzel.getChildren().add(knopf);  
  
        primaryStage.setScene(new Scene(wurzel, 300, 250));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

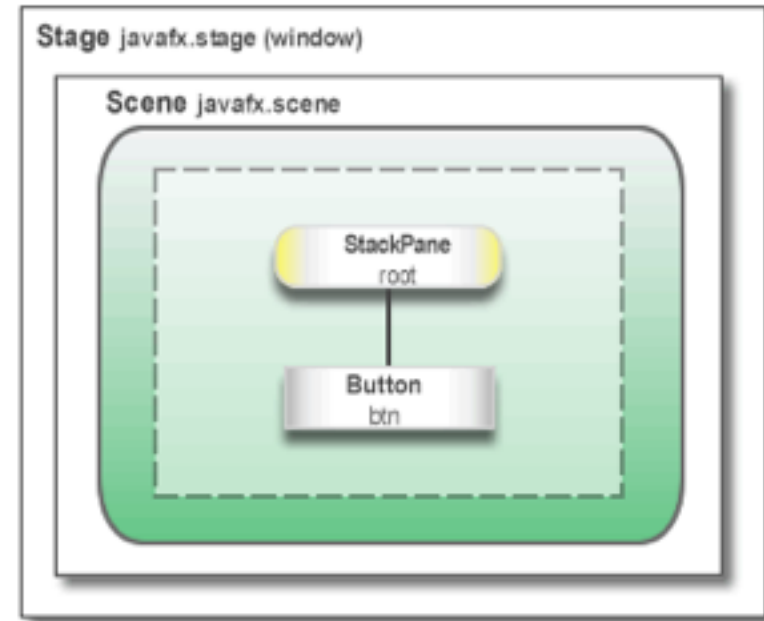
Fenster mit Knopf

- Startklasse einer JavaFX-Anwendung
 - erbt von `javafx.application.Application`
 - Methode `start()` ist der Einstiegspunkt
 - wird überschrieben
- Starten der Anwendung mit der statischen Methode `Application.launch()`

```
public class FensterMitKnopf extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Fenster mit Knopf!");  
  
        ToggleButton knopf = new ToggleButton();  
        knopf.setText("Ich bin ein Knopf!");  
        StackPane wurzel = new StackPane();  
        wurzel.getChildren().add(knopf);  
  
        primaryStage.setScene(new Scene(wurzel, 300, 250));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

Fenster mit Knopf

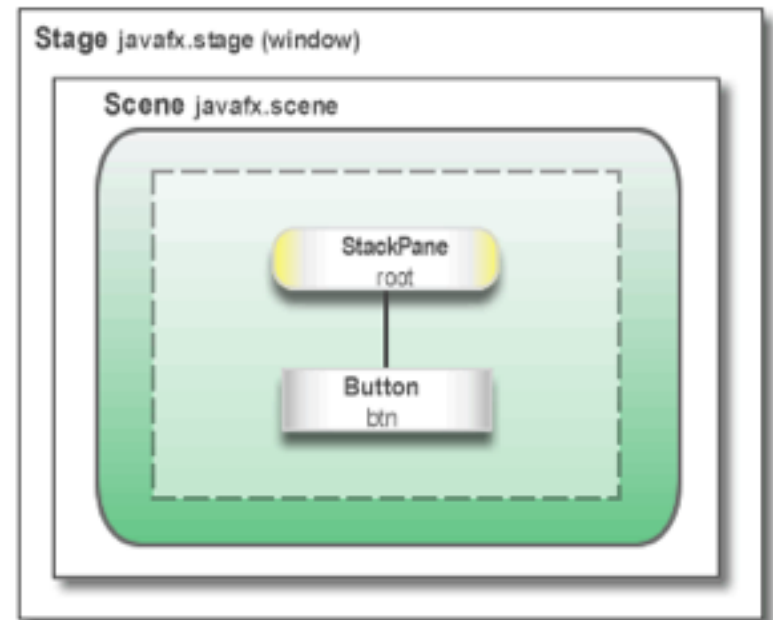
- Grafische Benutzerschnittstelle
 - engl. graphical user interface (GUI)
 - oberster Container: `javafx.stage`
- alle Komponenten der GUI
 - als Knoten in einem Szenengraph
 - eigentlich: eine Szenenbaum
 - oberster Knoten: Wurzelknoten (root,
 - Szenengraph befinden sich in der Szene (`javafx.scene`)



*Oracle JavaFX Dokumentation,
<http://docs.oracle.com/javase/8/javafx>,
abgerufen am 3.4.2014*

Fenster mit Knopf

- Beispiel: Fenster mit Knopf
 - Fenster durch Stage bereitgestellt
 - Knopf einziger Knoten im Szenengraph



*Oracle JavaFX Dokumentation,
<http://docs.oracle.com/javase/8/javafx>,
abgerufen am 3.4.2014*

Übung: Szenengraph

- Zeichnen Sie den Szenengraph für diesen Roboter bestehend aus
 - Torso
 - linker Arm
 - rechter Arm
 - linkes Bein
 - rechtes Bein
 - Kopf
- An welchem Knoten müssen Sie eine Eigenschaft (z.B. Farbe) setzen, damit sie für beide Beine gilt?



Quelle: Sony Qrio



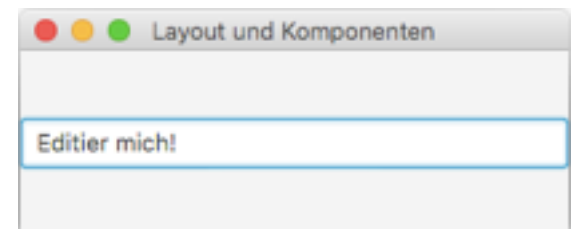
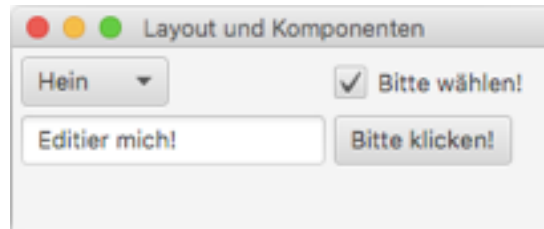
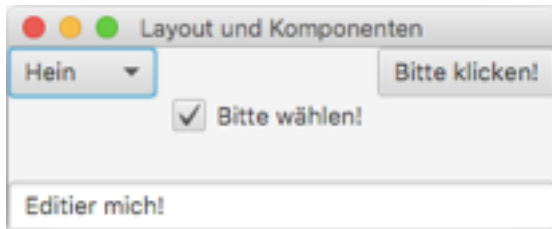
Anordnen von Komponenten

Zum Nachlesen:

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

Anordnen von Komponenten

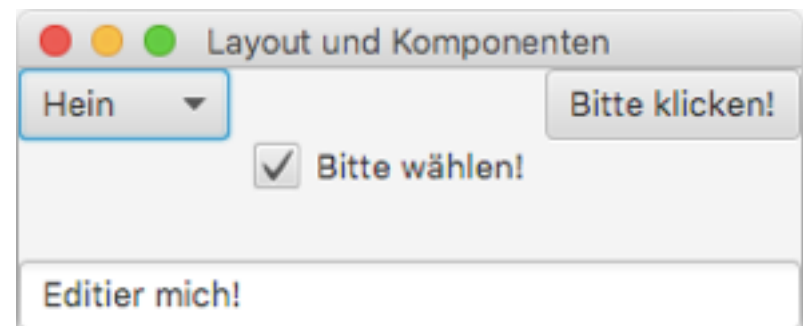
- Mehrere Komponenten sollen in einer Struktur gezielt angeordnet werden.



Anordnung: Richtungen

- Anordnung nach vordefinierten Richtungen: BorderLayout

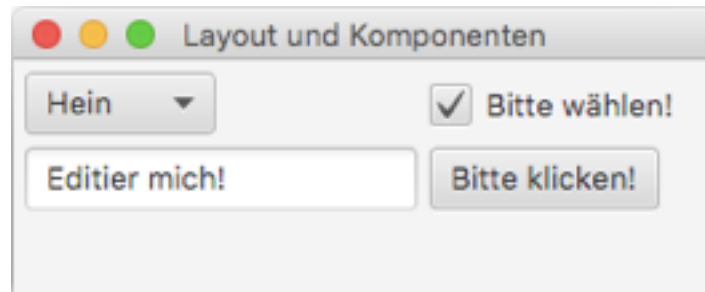
```
Pane wurzel = new BorderLayout();  
wurzel.setLeft(comboBox);  
wurzel.setCenter(auswahlbox);  
wurzel.setRight(knopf);  
wurzel.setBottom(textfeld);
```



Anordnung: Gitter

- Anordnung in einem Gitter: GridPane

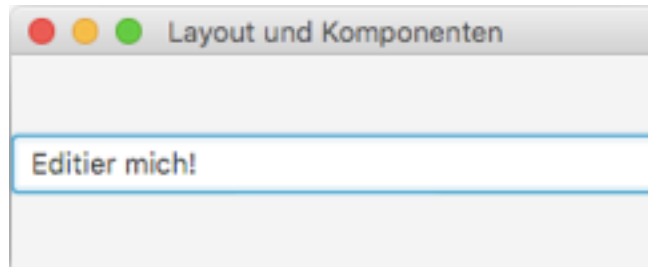
```
Pane wurzel = new GridPane();  
wurzel.add(comboBox, 0, 0);  
wurzel.add(auswahlbox, 0, 1);  
wurzel.add(knopf, 1, 0);  
wurzel.add(textfeld, 1, 1);
```



Anordnung: Stapel

- Anordnung in einem Stapel (übereinander): StackPane

```
Pane wurzel = new StackPane();  
wurzel.getChildren().add(comboBox);  
wurzel.getChildren().add(auswahlbox);  
wurzel.getChildren().add(knopf);  
wurzel.getChildren().add(textfeld);
```



Alternative Layouts

- AnchorPane
- FlowPane
- HBox
- TilePane
- VBox

Auswahlliste

`javafx.scene.control.ComboBox`

- Auswahlliste mit mehreren Optionen



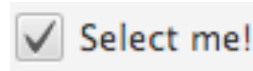
```
ComboBox<String> comboBox = new ComboBox<String>(
    FXCollections.observableArrayList("Jan", "Hein",
        "Klaas", "Pit"));
comboBox.setValue("Hein");
```

← JavaFX hat zusätzliche
Collections-Klassen

Checkbox

`javafx.scene.control.CheckBox`

- Binäre Auswahl: wahr vs. falsch



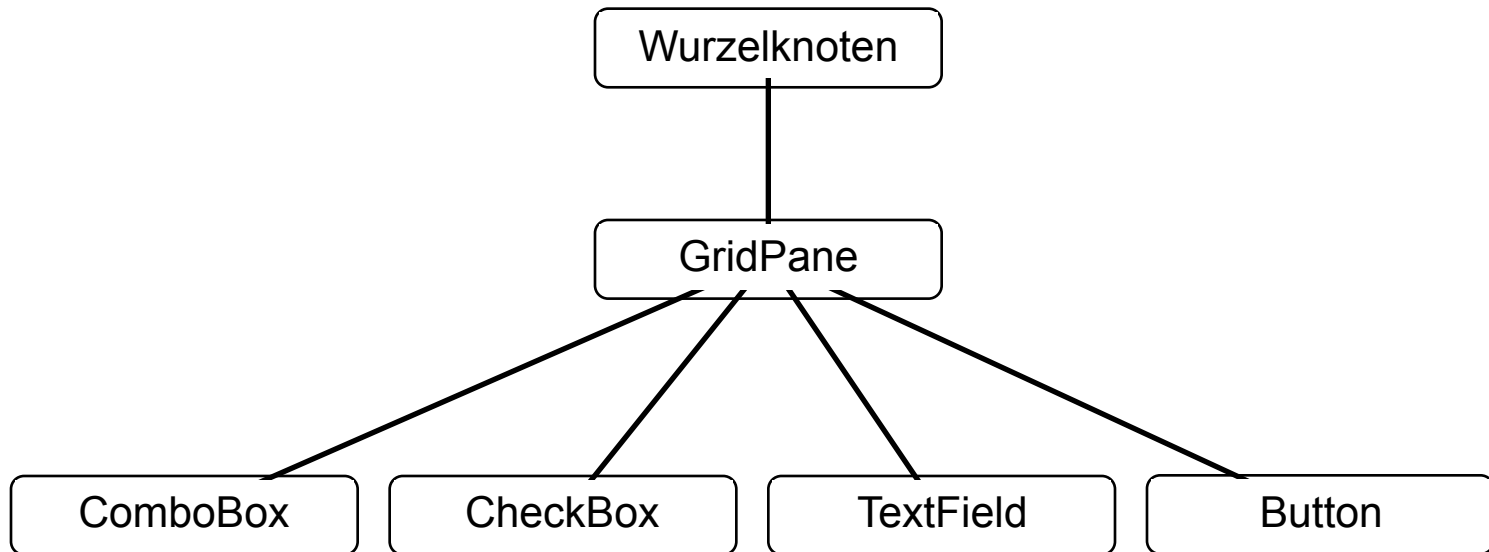
```
CheckBox auswahlbox = new CheckBox("Bitte wählen!");  
auswahlbox.setSelected(true);
```

Textfeld und Knopf

- Eingabe-Feld für Textzeilen (javafx.scene.control.TextField)
`TextField textfeld = new TextField("Editier mich!");`
- Knopf (javafx.scene.control.Button)
`Button knopf = new Button("Bitte klicken!");`
- viele weitere Komponenten vorhanden ...

Anordnen von Komponenten

- Szenengraph





Properties

Zum Nachlesen:

*<http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm>
<http://openbook.rheinwerk-verlag.de/javainsel9/> (Kapitel 10.3)*

Properties

- Objektvariablen (modifizierbare) Attribute eines Objektes
- Zugriff durch Konvention sichergestellt

```
public class Rechnung {  
    // Variable, die Property repräsentiert  
    private DoubleProperty betrag = new SimpleDoubleProperty();  
  
    // Lesender Zugriff: Getter  
    public final double getBetrag() {  
        return betrag.get();  
    }  
  
    // Schreibender Zugriff: Setter  
    public final void setBetrag(double value) {  
        betrag.set(value);  
    }  
  
    // Zugriff auf Property-Objekt  
    public DoubleProperty betragProperty() {  
        return betrag;  
    }  
}
```

Properties

- Was bringt das?
- Zugriff durch Name der Property
- Verbindung mit Property herstellen
 - Beobachter-Muster
 - auf Veränderungen reagieren

Listener für Veränderung

- Konstruktor von Rechnung ergänzen

```
public Rechnung(ChangeListener<Number> amountVeraenderungsListener){  
    betrag.addListener(amountVeraenderungsListener);  
}
```

- Listener implementieren

```
public class RechnungswertVeraenderung implements ChangeListener<Number> {  
    @Override  
    public void changed(ObservableValue<? extends Number> observable,  
        Number alterWert, Number neuerWert) {  
        System.out.println("Veränderung: " + alterWert + " -> " + neuerWert);  
    }  
}
```

- Veränderung wird beobachtet

```
RechnungswertVeraenderung veraenderung = new RechnungswertVeraenderung();  
Rechnung rechnung = new Rechnung(veraenderung);  
rechnung.setBetrag(23.42);
```

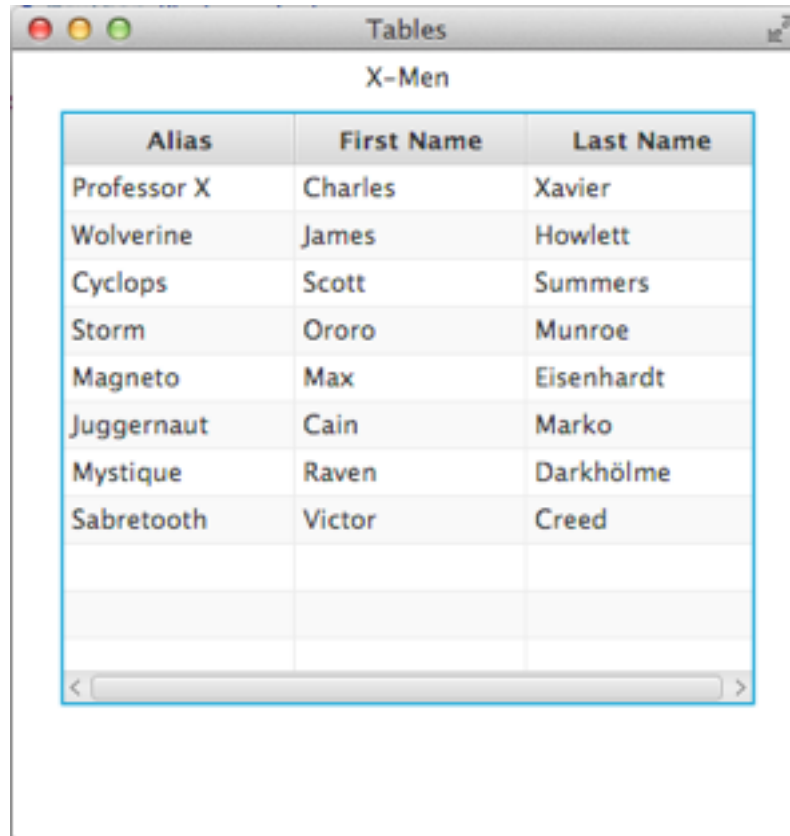
"Veränderung: 0.0 -> 23.42"



Tabellen

Tabellen

- Daten sollen tabellarisch angezeigt werden.



The screenshot shows a web browser window with the title 'Tables'. Inside the window, there is a table titled 'X-Men'. The table has three columns: 'Alias', 'First Name', and 'Last Name'. The data rows are as follows:

Alias	First Name	Last Name
Professor X	Charles	Xavier
Wolverine	James	Howlett
Cyclops	Scott	Summers
Storm	Ororo	Munroe
Magneto	Max	Eisenhardt
Juggernaut	Cain	Marko
Mystique	Raven	Darkhölme
Sabretooth	Victor	Creed

Below the table, there is a horizontal scrollbar with left and right arrow buttons.

Tabellen

- **Datenmodell**

- Darstellung von Personen
- Vorname, Nachname, Alias
 - z.B. "Charles", "Xavier", "Professor X"

- Synchronisation

- häufige Anforderungen: GUI-Element und Variable müssen synchronisiert werden
- hier: Datenmodell ändert sich → Tabelle aktualisieren

- Konzept dazu in JavaFX: `ObservableList<T>`

- wie Beobachter-Entwurfsmuster (Observer)
 - kommt im Kapitel: Entwurfsmuster
- Objekt "meldet sich", wenn es verändert wurde
- "Interessenten" können Ereignis abfangen

Daten

- Datenstruktur Person als Beispiel

```
public class Person {  
    private StringProperty vorname;  
    private StringProperty alias;  
    private StringProperty nachname;  
    ...  
}
```

- Liste der Personen wird in JavaFX-Collection (ObservableList<T>) verwaltet

```
ObservableList<Person> personen = FXCollections.<Person> observableArrayList();  
personen.add(new Person("Professor X", "Charles", "Xavier"));  
personen.add(new Person("Wolverine", "James", "Howlett"));  
personen.add(new Person("Cyclops", "Scott", "Summers"));  
[...]
```

Tabellenansicht

- Anlegen einer Tabellen-Darstellung (`javafx.scene.control.TableView`)
- Erzeugen einer Tabellenansicht

```
TableView<Person> tabellenAnsicht = new TableView<>( );
```

Spalten

- Tabelle wird aus Spalten(`javafx.scene.control.TableColumn`) zusammengesetzt

```
// Argument = Titel der Spalte
TableColumn<Person, String> aliasSpalte = new
    TableColumn<>( "Alias" );
aliasSpalte.setCellValueFactory( new PropertyValueFactory<Person,
    String>( „alias" ) );
```

- CellValueFactory:
 - Wie hole ich aus der Liste der Daten (hier: Personen) und der Zeilennummer die darzustellende Information?
- PropertyValueFactory
 - Verwende von den Daten-Objekt (hier: Person) die angegebene Property

Zusammensetzen

- letzter Schritt: Tabelle aus Spalten zusammensetzen
`tabellenAnsicht.setColumns().add(aliasSpalte);`

Hinweis: falls nur eine Spalte, korrekte Formatierung mit:

```
tabellenAnsicht.setColumnResizePolicy(  
    TableView.CONSTRAINED_RESIZE_POLICY );
```

Übung: Tabellen

- Nennen Sie die notwendigen Schritte, um die folgende Liste von Strings in einer JavaFX-Tabelle anzuzeigen?

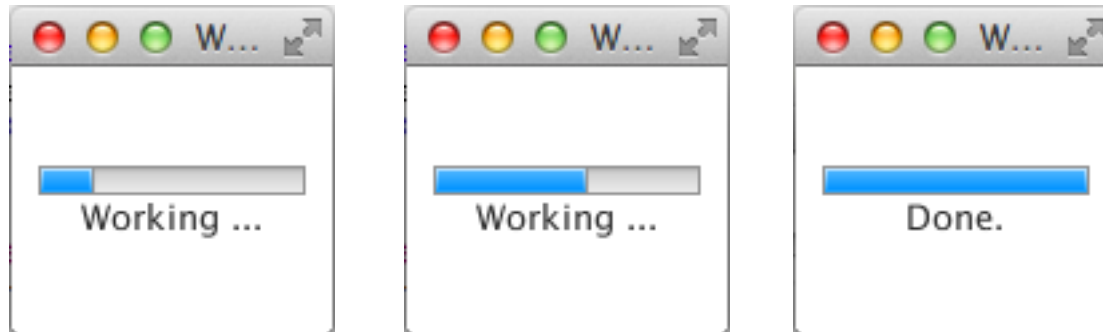
Marke	Typ
BMW	Mini One
Audi	Quattro
Mercedes	C-Klasse
Renault	Kangoo



Worker-Threads

Worker Threads

- Threads sollen Komponenten aus anderen Threads verändern.



JavaFX-Task

- Task verhält sich wie Runnable
- aber: Abbrechen mit cancel()
 - nicht interrupt()
- Thread-Operationen an JavaFX angepasst
- run()-Methode heißt hier call()

```
public class FortschrittsanzeigeTask extends
    Task<Boolean> {

    private final Label label;

    public FortschrittsanzeigeTask(Label label) {
        this.label = label;
    }

    @Override
    protected Boolean call() throws Exception {
        // Fortschritt aktualisieren
        int numberOfSteps = 20;
        updateProgress(0, numberOfSteps - 1);
        for (int i = 0; i < numberOfSteps; i++) {
            // Pausierne (zu Demozwecken)
            Thread.sleep(200);
            updateProgress(i, numberOfSteps - 1);
        }
        // Label aktualisieren
        updateLabel();
        // Alles ok!
        return true;
    }

    private void updateLabel() {
        Platform.runLater(() -> {
            label.setText("Done.");
        });
    }
}
```


Worker Threads

- jeder Task kann seinen Fortschritt verwalten
updateProgress(<aktueller Stand>, <Stand am Ende>);

```
protected Boolean call() throws Exception {  
    // Fortschritt aktualisieren  
    int numberOfSteps = 20;  
    updateProgress(0, numberOfSteps - 1);  
    for (int i = 0; i < numberOfSteps; i++) {  
        // Pausierne (zu Demozwecken)  
        Thread.sleep(200);  
        updateProgress(i, numberOfSteps - 1);  
    }  
    // Label aktualisieren  
    updateLabel();  
    // Alles ok!  
    return true;  
}
```

Update des Labels

- Ende der `call()`-Methode in `SampleTask`: `updateLabel()`
- Achtung: GUI-Komponenten von JavaFX nicht Thread-sicher!
- Veränderungen an der GUI nur durch FX-Task
- Aufgabe wird als `Runnable` and `JavaFX-Task` übergeben (hier als Lambda)

```
Platform.runLater(( ) -> {  
    label.setText( "Done." );  
});
```
- Achtung: nur der `JavaFX-Task` darf Änderung an GUI-Komponenten vornehmen
- ansonsten können Synchronisationsprobleme auftreten

Starten

```
// Fortschrittsanzeige
ProgressBar fortschrittsanzeige = new ProgressBar();
wurzel.getChildren().add(fortschrittsanzeige);

// Überschrift
Label label = new Label("in Arbeit ...");
wurzel.getChildren().add(label);

// Dieser Task aktualisiert die Fortschrittsanzeige
FortschrittsanzeigeTask task = new FortschrittsanzeigeTask(label);

// Task wird an die Fortschrittsanzeige gebunden (vorher alle alten
// Bindungen auflösen)
fortschrittsanzeige.progressProperty().unbind();
fortschrittsanzeige.progressProperty().bind(task.progressProperty());

// Task starten
Thread worker = new Thread(task);
worker.start();
```



Scene Builder

Zum Nachlesen:

<http://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/>

Voraussetzungen

- Visuelles Erstellen einer Grafischen Benutzeroberfläche
- Entwicklung unter Eclipse: e(fx)clipse-Plugin installieren:
 - *<http://download.eclipse.org/efxclipse/updates-released/2.3.0/site>*
 - *(oder neuer)*

Konzept

- Trennung von
 - Layout der Komponenten (XML)
 - Stil (Cascading Style Sheets, CSS)
 - Code (Controller)



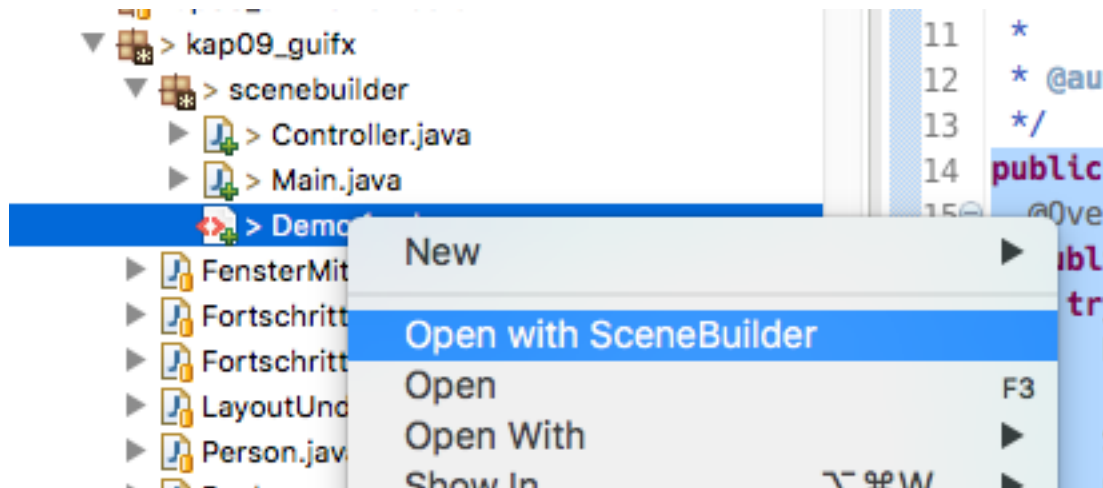
Loslegen

- Repräsentation des Layouts: *.fxml
 - New → Other → JavaFX → New FXML Document
- eigentliche Anwendung = reguläre Java-Klasse mit main()-Methode
 - meist: Main.java
- Code (z.B. für Ereignisbehandlung):
 - Controller.java

JavaFX-Anwendungsklasse (Beispiel)

```
public class Main extends Application {  
    @Override  
    public void start(Stage primaryStage) {  
        try {  
            BorderPane root =  
                (BorderPane) FXMLLoader.load(getClass()  
                    .getResource(<FXML-Dateiname>)); //z.B. Layout.fxml  
            Scene scene = new Scene(root, 400, 400);  
            primaryStage.setScene(scene);  
            primaryStage.show();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

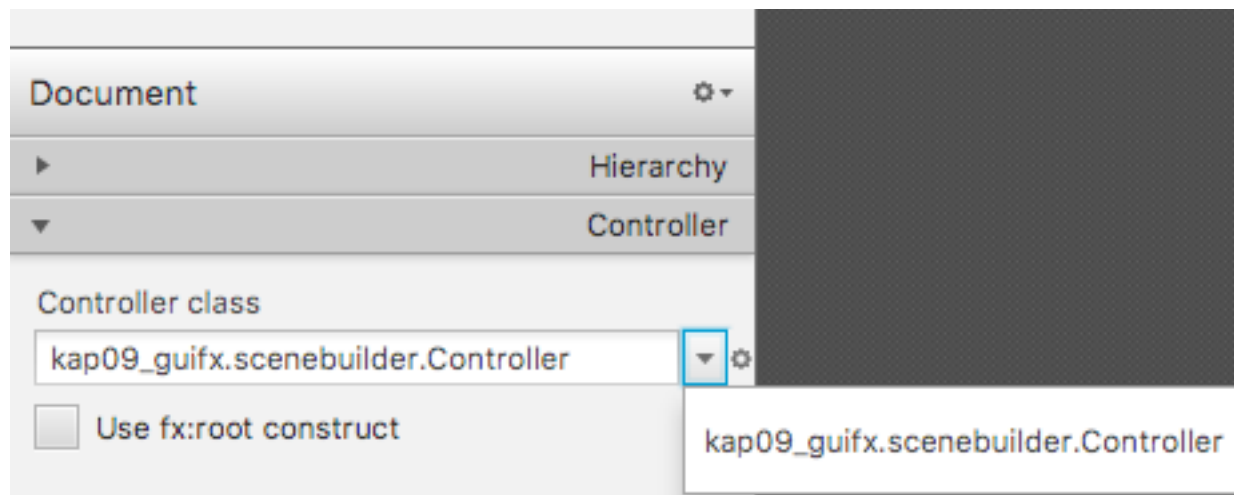

Scene Builder Verwenden



Quellcode vs. FXML: Controller

- Programmierung über Java-Klasse

```
public class Controller implements Initializable {  
    ...  
    @Override  
    public void initialize(URL location, ResourceBundle resources) {  
        // Initialisierung  
        ...  
    }  
    ...  
}
```



Komponenten Layouten

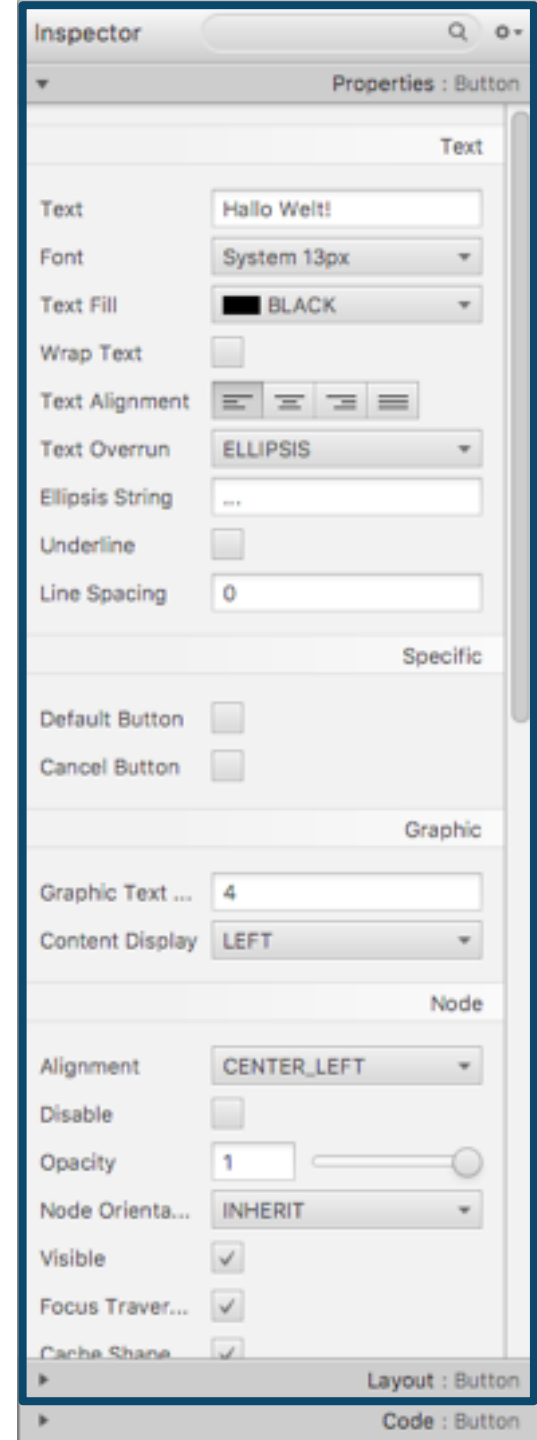
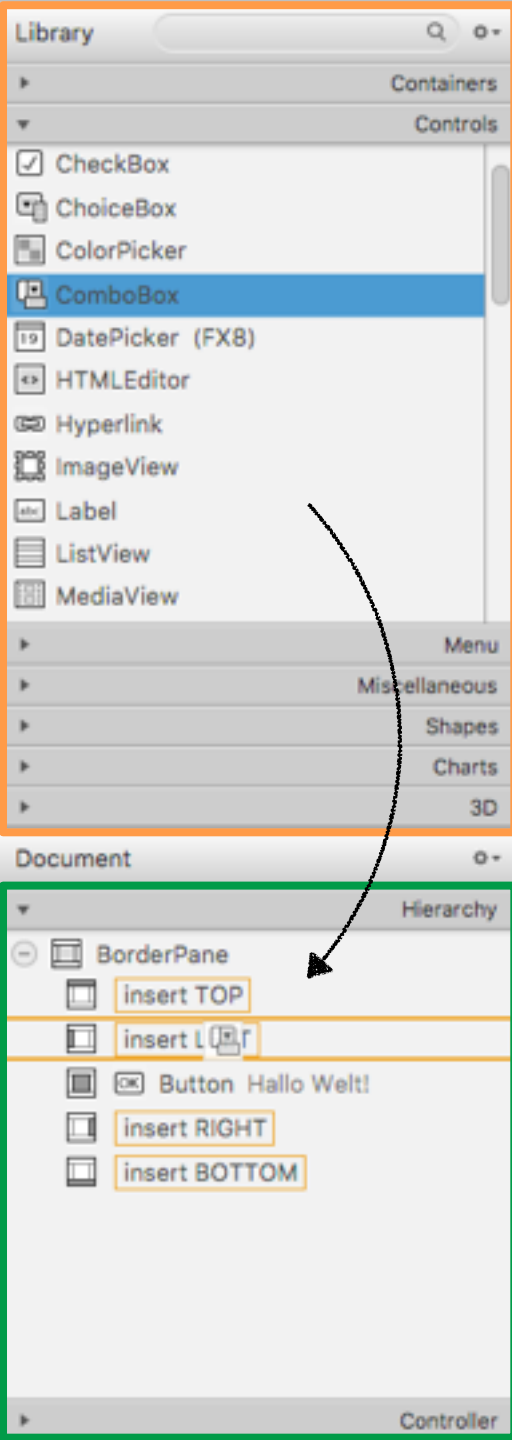
Bibliothek

Inspektor

- Per Drag'n'Drop aus der Bibliothek in den Szenengraphen
- Eigenschaften der Komponenten im Inspektor setzen

Szenengraph

formatik



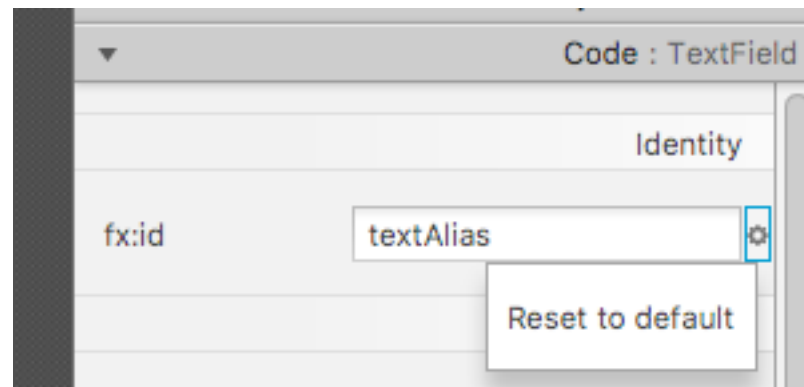
Verbindung Komponente-Quellcode

- Im Controller (Annotation + Deklaration)

@FXML

private TextField textAlias;

- Im Scene Builder (Inspector):
 - Id (= Bezeichner) setzen



Tabellen mit Scene Builder

- TableView und TableColumn im Scene Builder modellieren
- CellValueFactory setzen

- entweder manuell im XML

```
<TableColumn prefWidth="75.0" text="Alias">  
    <cellValueFactory><PropertyValueFactory property="alias" /></cellValueFactory>  
</TableColumn>
```

- oder im Controller

```
spalteAlias.setCellValueFactory(  
    new PropertyValueFactory<Person, String>(„alias“));
```

- Daten im Controller anbinden

```
tabelle.setItems(Datensatz.getPersonenDatensatz());
```

Zusammenfassung

- Grafische Benutzeroberflächen mit JavaFX
- Fenster mit Knopf
- Anordnen von Komponenten
- Properties
- Tabellen
- Worker Threads
- Scene Builder