Java FX 8 Workshop

Michael Inden

Speaker – Kurzlebenslauf

zühlke
empowering ideas

- Michael Inden, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg
- ~9 Jahre bei Heidelberger Druckmaschinen AG in Kiel
- ~7 Jahre bei IVU Traffic Technologies AG in Aachen
- Seit 2013 bei Zühlke Engineering AG in Zürich
  (We are hiring …)
- Autor und Gutachter
  beim dpunkt.verlag

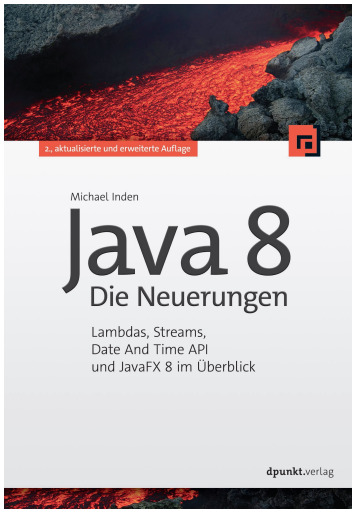


Table of Contents

zühlke
empowering ideas

zühlke
empowering ideas

Part 1: Grundlagen

Grundlagen, Bedienelemente und CSS-Styling

Michael Inden

JavaFX im Überblick

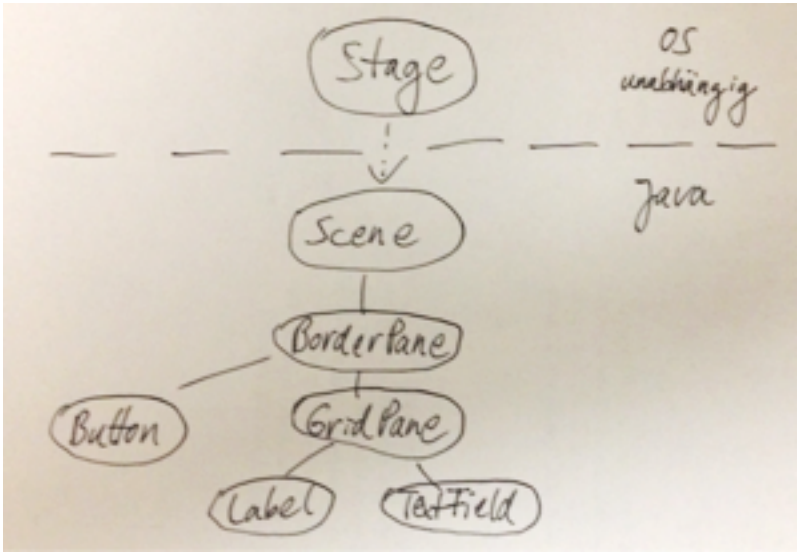
zühlke
empowering ideas

- JavaFX als Nachfolger von Swing
- Desktop-GUI-Programmierung erleichtern / attraktive GUIs
- hoher Bedienkomfort unterstützt durch Effekte und Animationen
- Styling mit CSS, deklarative GUIs mit FXML
- Data-Binding
- Diverse Bedienelemente:
  - Label, Button, TextField, ...
  - TabView, ListView, TableView, TreeTableView
  - Charts
- Unterstützung von 3D

Grundbegriffe

zühlke
empowering ideas

- **Stage** - Die Bühne bildet die Abstraktion zum Betriebssystem
- **Scene** - Die zentrale Containerkomponenten auf der alle anderen Nodes platziert werden.
- **Nodes** - Die Nodes bilden den Scenegraph und getScene() liefert die Scene zu einer Node

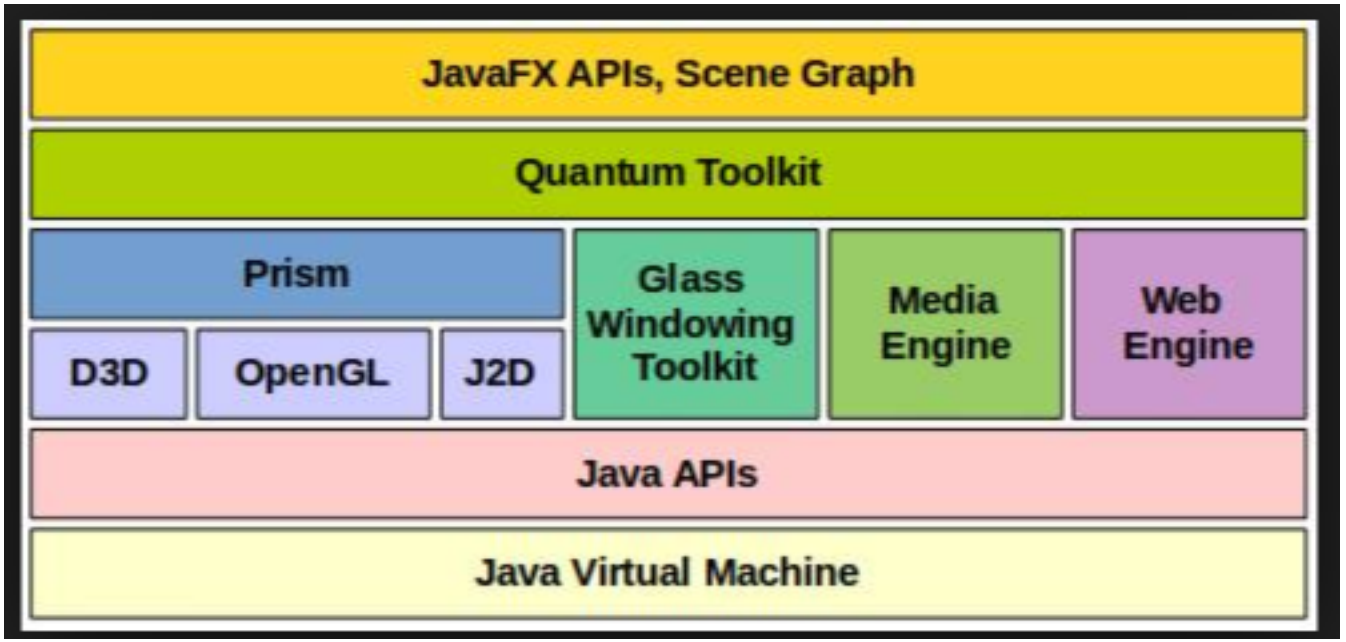


- (Fast) alles auf dem Bildschirm ist eine Node

Architektur

zühlke
empowering ideas

- Eigener Renderer-Thread => sorgt für 60 FPS
- Kein aktives Zeichnen (Repaint) nötig, Änderungen werden sofort sichtbar
- Nutzt alternativ Direct Draw, Open GL, Java 2D



HelloWorld mit JavaFX

zühlke
empowering ideas

- `start(Stage)`-Methode: man erzeugt nur die Elemente des SceneGraph, nicht aber die Stage => kein `new JFrame()`

```
@Override
public void start(final Stage stage) throws Exception
{
    final StackPane stackPane = new StackPane();

    final Node labelNode = new Label("Hello Java FX World!");
    stackPane.getChildren().add(labelNode);

    // Stage und Scene verbinden
    stage.setScene(new Scene(stackPane, 250, 75));

    // Titel und Resizable-Eigenschaft setzen
    stage.setTitle("FirstJavaFxExample");
    stage.setResizable(true);

    // Positionierung und Sichtbarkeit
    stage.centerOnScreen();
    stage.show();
}
```

HelloWorld mit JavaFX

zühlke
empowering ideas

- Basisklasse `Application`

```
public class FirstJavaFxExample extends Application
{
    @Override
    public void start(final Stage stage) throws Exception
    {
        // ...
    }

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

Action Handling


```
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 event)
        {
                pane.getChildren().add(new Label("- Hello World! -"));
        }
});
```

Einfacher mit Lambda:

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

Layout-Management

zühlke
empowering ideas

- Ähnlich zu Swing, statt Layoutmanager gibt es aber analog folgende spezielle Container: `StackPane`, `FlowPane`
- HBox, VBox, `GridPane`, BorderPane

```
@Override
public void start(final Stage primaryStage)
{
    final BorderPane borderPane = new BorderPane();
    borderPane.setPadding(new Insets(7,7,7,7));
    borderPane.setTop(createToolbarPane());
    borderPane.setCenter(createInputPane());
    borderPane.setLeft(createNavigationPane());

    primaryStage.setTitle(LayoutCombinationExample.class.getSimpleName());
    primaryStage.setScene(new Scene(borderPane, 350, 250));
    primaryStage.show();
}
```

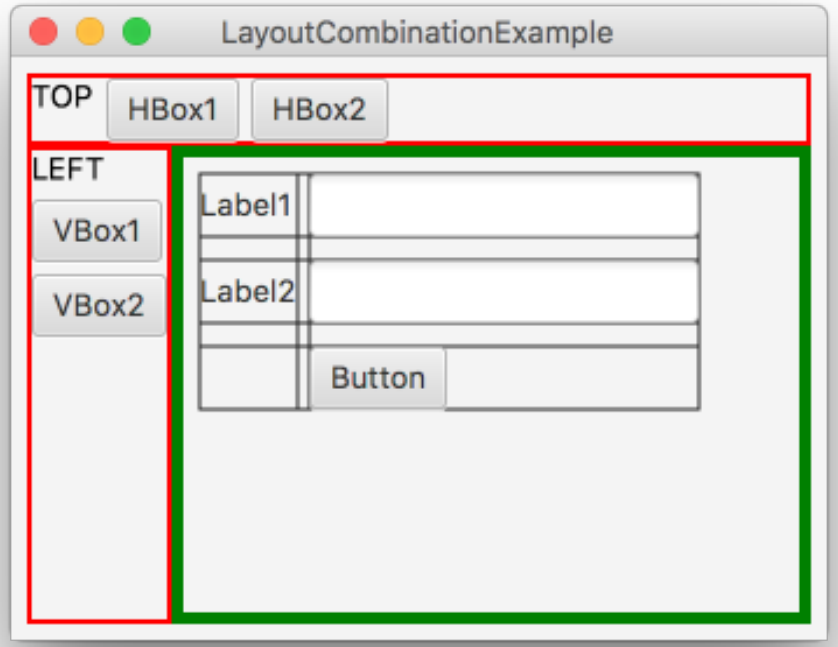
- Tipp: GUI nicht komplett in `start()` konstruieren

Layout-Management

zühlke
empowering ideas

- **HBox und VBox**

```
private Pane createToolbarPane()
{
    final HBox hbox = new HBox(5);
    hbox.setStyle("-fx-border-width: 2;-fx-border-color: red;");
    hbox.getChildren().addAll(new Text("TOP"), new Button("HBox1"),
                              new Button("HBox2"));
    return hbox;
}
```



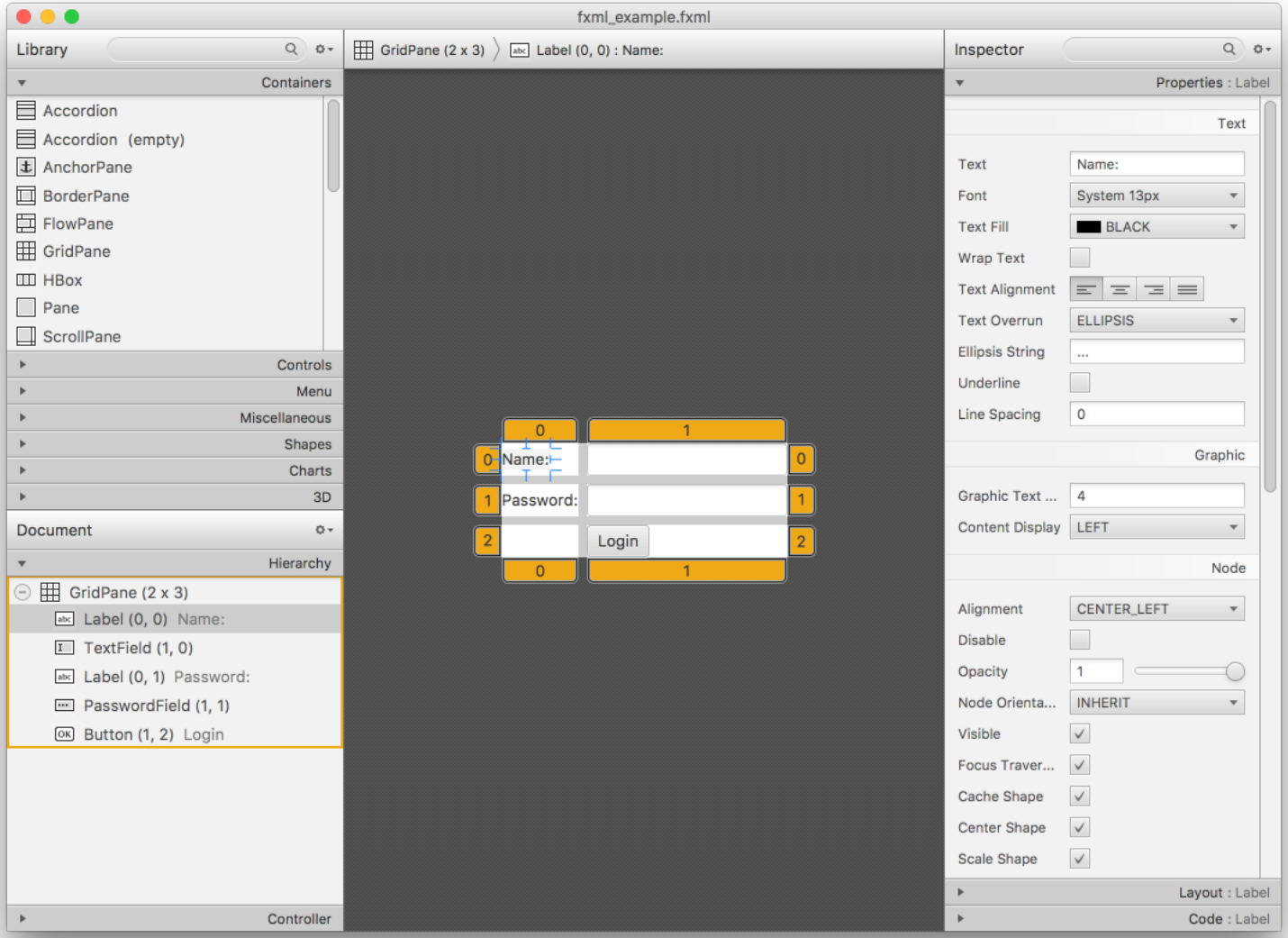
Layout mit GridPane

zühlke
empowering ideas

```
private Pane createInputPane()
{
    final GridPane gridPane = new GridPane();
    gridPane.setBorder(new Border(new BorderStroke(Color.GREEN,
                       BorderStrokeStyle.SOLID, null, new BorderWidths(5))));
    gridPane.setPadding(new Insets(7,7,7,7));
    gridPane.setGridLinesVisible(true);
    gridPane.setHgap(5);
    gridPane.setVgap(10);

    final Label label1 = new Label("Label1");
    final TextField textfield1 = new TextField();
    final Label label2 = new Label("Label2");
    final TextField textfield2 = new TextField();
    final Button button = new Button("Button");
    gridPane.add(label1, 0, 0);
    gridPane.add(textfield1, 1, 0);
    gridPane.add(label2, 0, 1);
    gridPane.add(textfield2, 1, 1);
    gridPane.add(button, 1, 2);

    return gridPane;
}
```

SceneBuilder als GUI-Design-Tool

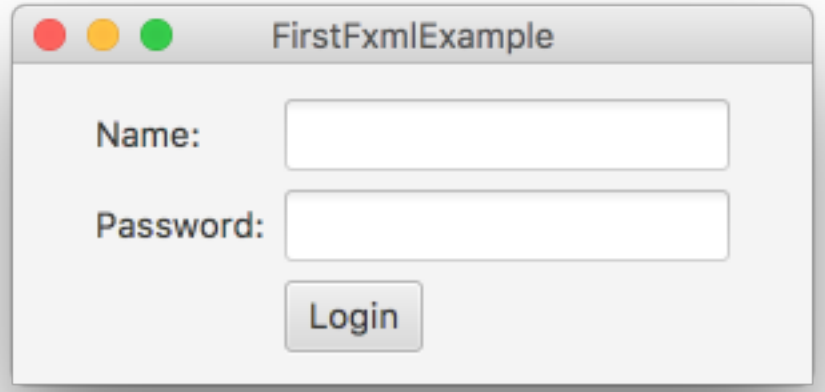
zühlke
empowering ideas



FXML und Darstellung

zühlke
empowering ideas

```
<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="chapter5_javafx.fxml.FXMLExampleController">

  <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>
```



Deklarative Beschreibung mit FXML

zühlke
empowering ideas

- FXML laden:

```
@Override
public void start(final Stage stage) throws Exception
{
        final Parent root = FXMLLoader.load(getClass().
                                getResource("login_example.fxml"));

        stage.setScene(new Scene(root, 400, 200));
        stage.setTitle("FirstFxmlExample");
        stage.show();
}
```

Controller & Aktionen im FXML

zühlke
empowering ideas

- **Spezielle Tags im FXML:**

```
<GridPane hgap="7.0" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="129.0"
prefWidth="319.0“ vgap="7.0"
xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/8.0.65"
fx:controller="fxmldemo.LoginController" >

...

<Button id="loginBtn" fx:id="loginBtn" mnemonicParsing="false"
                      onAction="#handleLoginButton" text="Login"
        GridPane.columnIndex="1"
        GridPane.rowIndex="2" />

...

</GridPane>
```

Controller & Aktionen in Java

zühlke
empowering ideas

```
public class LoginController
{
    @FXML
    private Button loginBtn;

    @FXML
    void handleLoginButton(ActionEvent event)
    {
       System.out.println("Login pressed");
    }
}
```

Die @FXML annotierten Elemente werden mit dem FXML-Dokument beim Laden verknüpft.

Rich-Client mit CSS

zühlke
empowering ideas

- In JavaFX lassen sich nahezu alle Dinge mithilfe von CSS optisch interessanter gestallten
- Spezielle -fx-Flags
  - `-fx-text-fill`
  - `-fx-background-color`
  - `-fx-font-size`
  - ...
- Gradienten
  - `linear-gradient`
  - `radial-gradient`

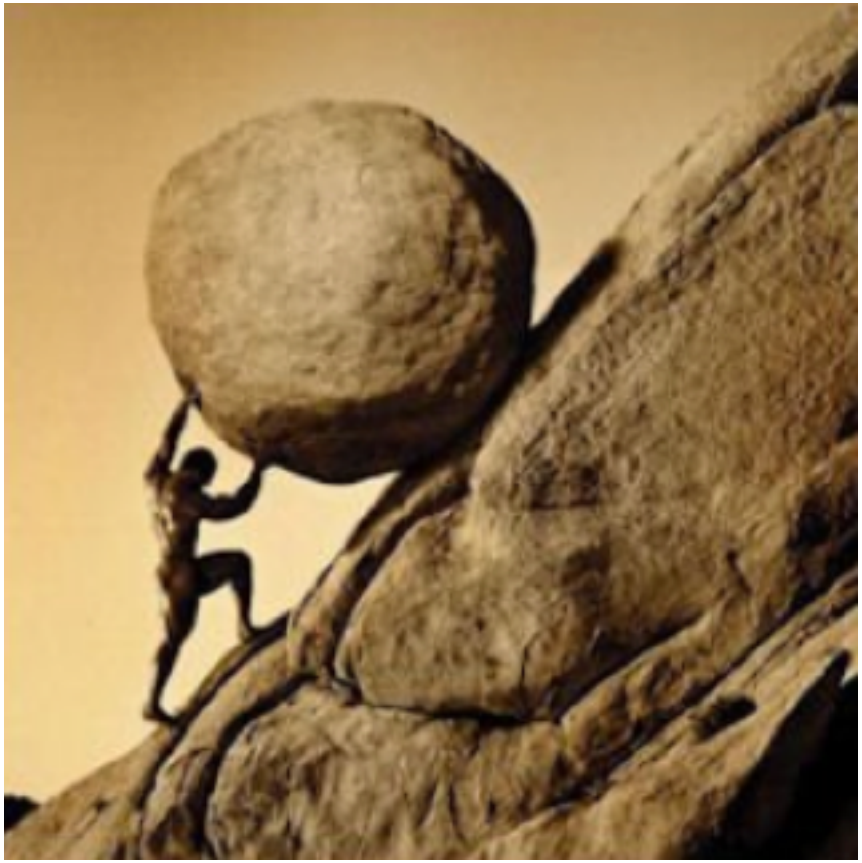
Rich-Client mit CSS

zühlke
empowering ideas

```
public void start(Stage primaryStage) throws Exception
{
    final Button loginbutton = new Button("Login Button");
    loginbutton.setStyle("-fx-text-fill: silver; -fx-font-size: 18pt;" +
                         "-fx-font-weight: bold; " + "-fx-background-color: " +
                         "radial-gradient(center 25% 25%, radius 50%, " +
                         "reflect, dodgerblue, darkblue 75%, dodgerblue);");
```


```
loginButton.getStyleClass().add("customloginbutton");
fancyButton.setId("fancybutton");
```

Übungen Teil 1



zühlke
empowering ideas

Part 2: Properties and more

- Properties und Bindings
- ObservableCollections
- Concurrency

Michael Inden

Properties

zühlke
empowering ideas

- **Properties: Attribute / Variablen mit Benachrichtigungsfunktionalität**

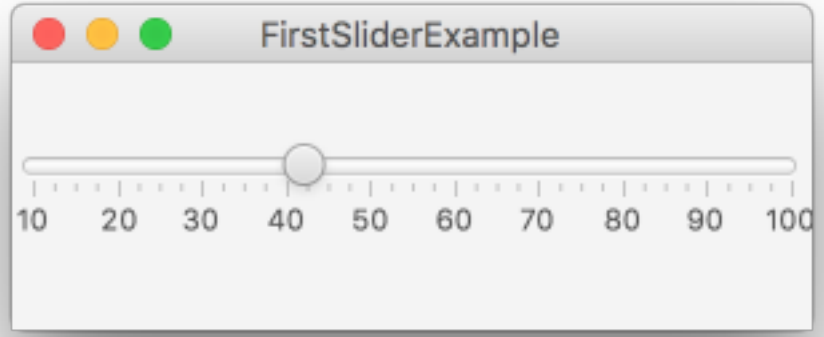
```
// Zugriffe
final StringProperty textProp = new SimpleStringProperty("MICHA");

System.out.println("textProp:    " + textProp);
System.out.println("getValue(): " + textProp.getValue());

// Berechnungen
final IntegerProperty intProp1 = new SimpleIntegerProperty(10);
final IntegerProperty intProp2 = new SimpleIntegerProperty(2);

System.out.println("subtract(): " + intProp1.add(40).subtract(intProp2));
System.out.println("multiply(): " + intProp1.multiply(intProp2).getValue());
```

```
textProp:    StringProperty [value: MICHA]
getValue(): MICHA
subtract(): IntegerBinding [invalid]
multiply(): 20
```

Properties + Controls

zühlke
empowering ideas

```
final Slider slider = new Slider(10, 100, 42.195);
slider.setShowTickMarks(true);
slider.setShowTickLabels(true);
slider.setMajorTickUnit(10);
slider.setBlockIncrement(10);
```

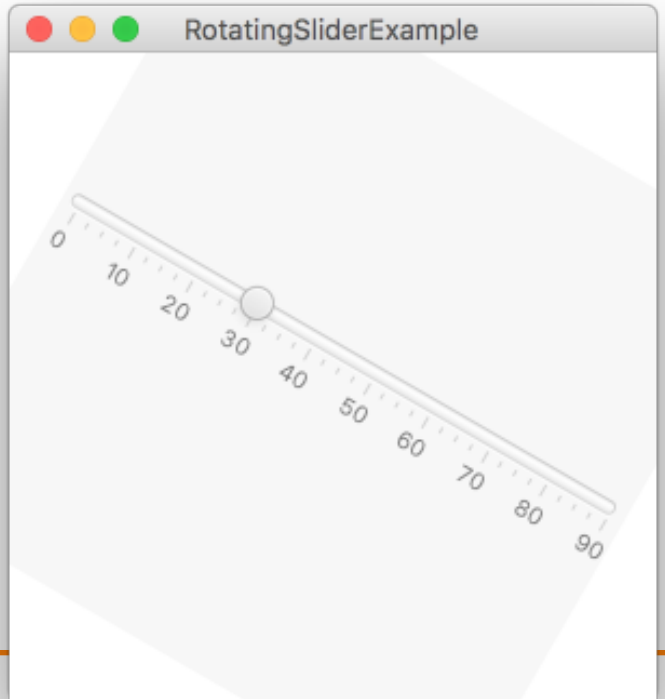


Properties + Controls + Binding

zühlke
empowering ideas

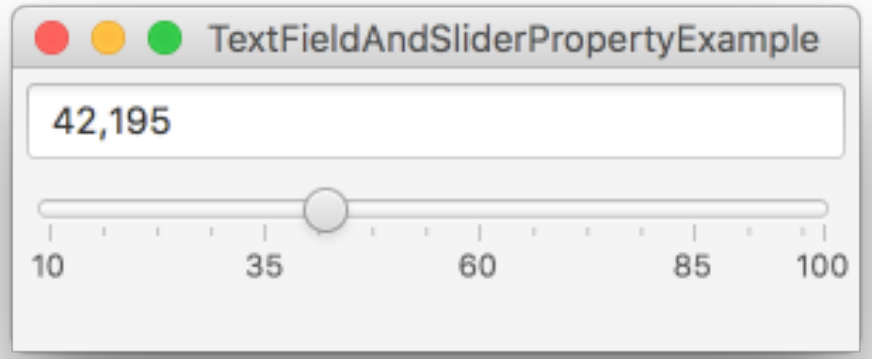
- Controls veröffentlichen Attribute als Properties:
  - `valueProperty() / rotateProperty() / ...`
- Bindung: Verknüpfung unterschiedlicher Properties miteinander

```
slider.rotateProperty().bind(slider.valueProperty());
```



Binding

- **Daten und GUI automatisch miteinander abgleichen**
  - Textfeldbreite und die Größe des Fensters
  - Textfeld und Slider
- **Bindings: Uni- / Bidirektional**
- **textField.textProperty().*bindBidirectional*(slider.valueProperty(), new NumberStringConverter());**



ObservableCollections

zühlke
empowering ideas

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

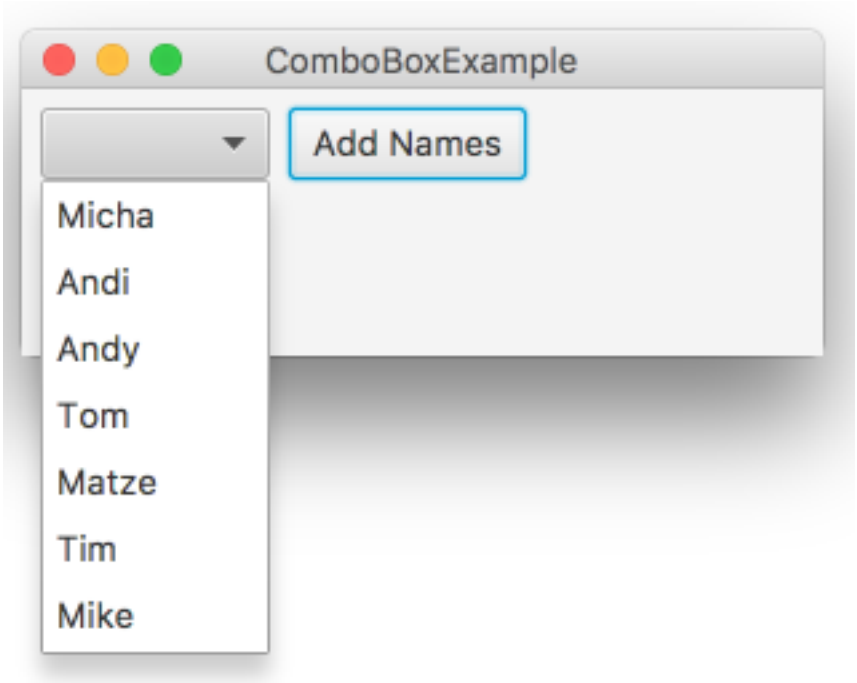
```
final String[] content = { "Orig1", "Orig2" }
final ObservableList<String> observableList =
                              FXCollections.observableArrayList(content);

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]
```

ObservableList + ComboBox

zühlke
empowering ideas

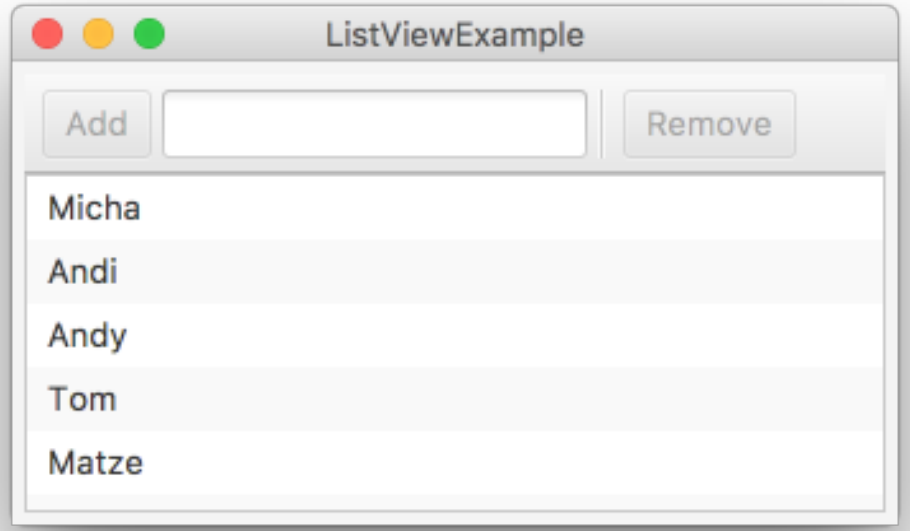
```
final String[] names = { "Micha", "Andi", "Andy", "Tom", "Matze" };
final ObservableList<String> entries = FXCollections.observableArrayList(names);

final ComboBox<String> comboNames = new ComboBox<>(entries);
final Button addNamesBtn = new Button("Add Names");
addNamesBtn.setOnAction(evt -> { entries.addAll("Tim", "Mike");
                                 comboNames.show(); });
```



ObservableList + EditableListView

zühlke
empowering ideas

```
final String[] names = { "Micha", "Andi", "Andy", "Tom", "Matze" };
final ObservableList<String> entries = FXCollections.observableArrayList(names);

final ListView<String> listView = new ListView<>(entries);
final SelectionModel<String> selectionModel = listView.getSelectionModel();
```




```
removeButton.disableProperty().bind(Bindings.isNull(
                                    selectionModel.selectedItemProperty()));
```

Concurrency mit JavaFX

zühlke
empowering ideas

- verschiedener Hilfsklassen zur Erleichterung, eher High-Level als Low-Level
- gute Integration mit den Properties und Observables
- Zentrale Bestandteile: Interface **Worker** und die Klassen **Task<V>** und **Service<V>**

- Die abstrakte Klasse `Task<V>` implementiert das Interface `Worker` und bietet u.a. folgende wichtige Methoden:
  - `abstract call()`
  - `cancel()`
  - `isCancelled() / isRunning() / isDone()`
  - `updateProgress(long workDone, long totalWork)`
  - `updateMessage(String)`

Concurrency mit JavaFX

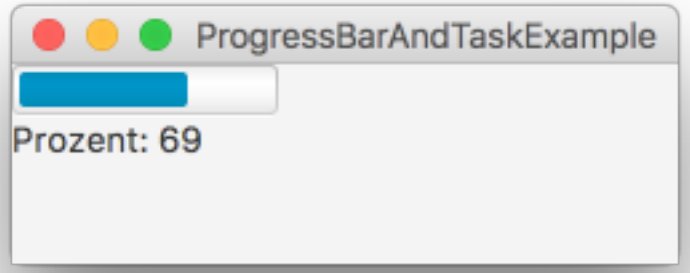
zühlke
empowering ideas

```
private Task<Void> createTask()
{
    final Task<Void> task = new Task<Void>()
    {
        @Override
        public Void call() throws InterruptedException
        {
            for (int i = 1; i <= 100; i++)
            {
                updateProgress(i, 100);
                updateMessage("Prozent: " + i);

                Thread.sleep(100);
            }
            return null;
        }
    };
    return task;
}
```

Concurrency mit JavaFX

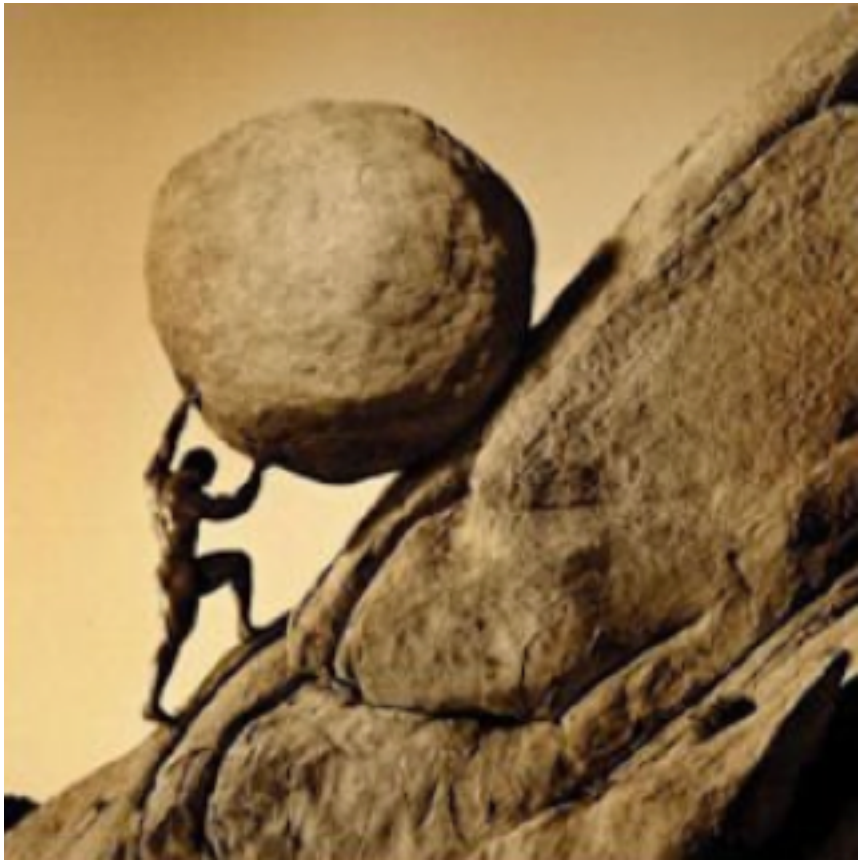
zühlke
empowering ideas

```
@Override
public void start(Stage primaryStage) throws Exception {

        final Task<Void> task = createTask();

        final ProgressBar bar = new ProgressBar();
        bar.progressProperty().bind(task.progressProperty());

        final Label info = new Label();
        info.textProperty().bind(task.messageProperty());

        new Thread(task).start();

        primaryStage.setScene(new Scene(new VBox(bar, info), 250, 75));
        primaryStage.setTitle("ProgressBarAndTaskExample");
        primaryStage.show();
}
```



Übungen Teil 2

zühlke
empowering ideas



zühlke
empowering ideas

Part 3: Tree, Table & TreeTable

Michael Inden

TreeItem

zühlke
empowering ideas

Zentrale Klassen: `TreeItem` definiert einen Knoten in einem Baum

```
final TreeItem<String> rootItem = new TreeItem<>("Root");
final TreeItem<String> groupA = new TreeItem<>("Group A");
final TreeItem<String> groupB = new TreeItem<>("Group B");
final TreeItem<String> itemA1 = new TreeItem<>("A1");
final TreeItem<String> itemA2 = new TreeItem<>("A2");
final TreeItem<String> itemB = new TreeItem<>("B");
```

Strukturierung durch Verknüpfung mit `ObservablList` children

```
rootItem.getChildren().add(groupA);
groupA.getChildren().addAll(itemA, itemA2);

...
```

Tree + TreeItem

zühlke
empowering ideas

```
final TreeItem<String> rootItem = new TreeItem<>("Root");
final TreeItem<String> groupA = new TreeItem<>("Group A");
final TreeItem<String> groupB = new TreeItem<>("Group B");
...
rootItem.getChildren().addAll(groupA, groupB);
groupA.getChildren().addAll(itemA1, itemA2);
...
final TreeView<String> treeView = new TreeView<>(treeItemRoot);
```

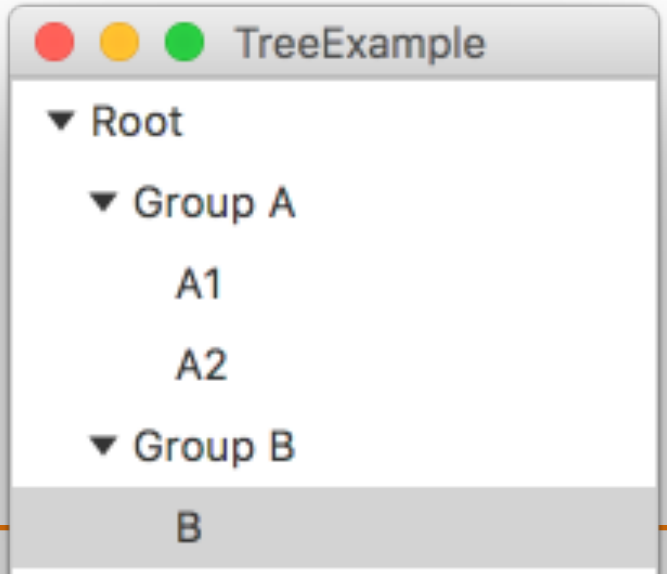


Table + TableColumn

zühlke
empowering ideas

Zentrale Klassen: `TableView` **und** `TableColumn`

```
TableView table = new TableView();

TableColumn firstNameCol = new TableColumn("First Name");
TableColumn lastNameCol = new TableColumn("Last Name");
TableColumn emailCol = new TableColumn("Email");

table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);
```

Table - Datenmodell

zühlke
empowering ideas

```
final ObservableList<Person> persons = FXCollections.observableArrayList(
            new Person("Mike", "Smith", "mike.smith@example.com"),
            new Person("Tim", "Bötz", "tb@tb.com"),
            new Person("Matze", "Man", "powerballs@kiel.de"),
            new Person("Michael", "Inden", "michael.inden@zuehlke.com")
        );

final TableView<Person> table = new TableView<>(persons);

final TableColumn<Person, String> firstNameCol = new TableColumn<>("First Name");
final TableColumn<Person, String> lastNameCol = new TableColumn<>("Last Name");
final TableColumn<Person, String> emailCol = new TableColumn<>("Email");
```

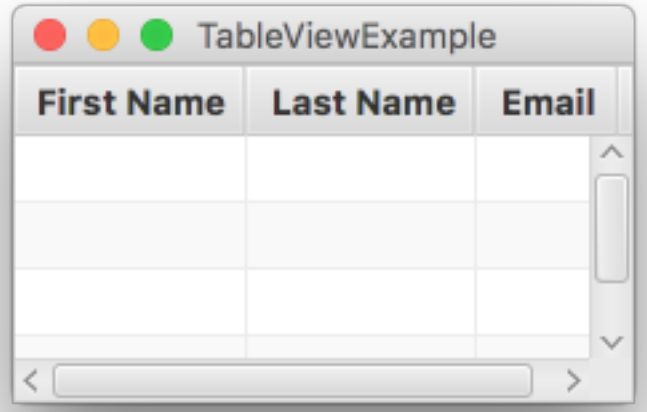


Table – Verknüpfung mit Daten

zühlke
empowering ideas

```
firstNameCol.setCellValueFactory(
        new PropertyValueFactory<Person, String>("firstName"));
lastNameCol.setCellValueFactory(
        new PropertyValueFactory<Person, String>("lastName"));
emailCol.setCellValueFactory(
        new PropertyValueFactory<Person, String>("email"));
```

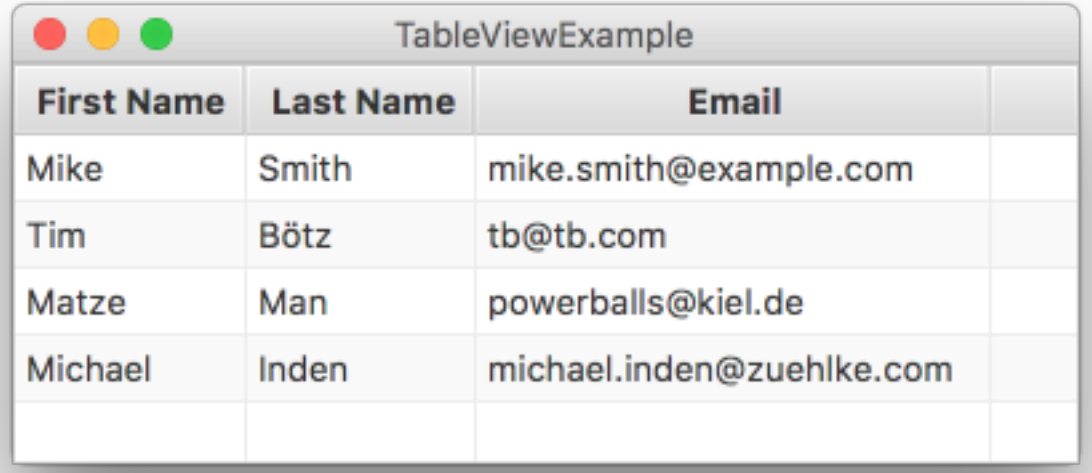


Table – Editierbarkeit

zühlke
empowering ideas

```
table.setEditable(true);

firstNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
firstNameCol.setOnEditCommit((CellEditEvent<Person, String> event) -> {
    // Old, conventional style
    final int row = event.getTablePosition().getRow();
    final Person person = (Person) event.getTableView().getItems().get(row);
    person.setFirstName(event.getNewValue());
});

lastNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
lastNameCol.setOnEditCommit(event -> {
    // Shortcut, more understandable way
    final Person person = event.getRowValue();
    person.setLastName(event.getNewValue());
});
```

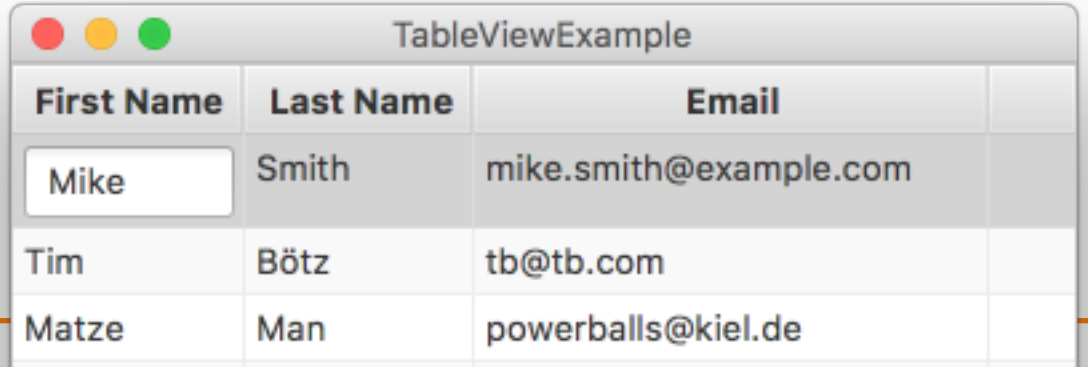
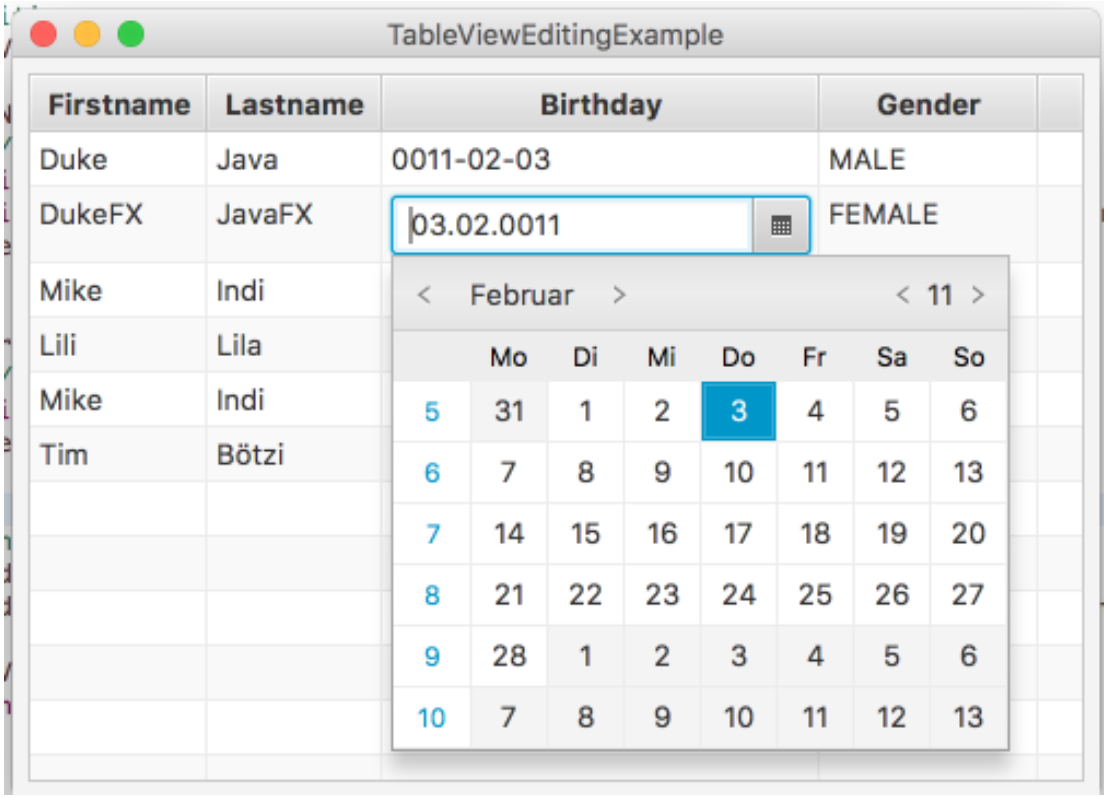


Table – Editierbarkeit mit eigenem Editor

zühlke
empowering ideas

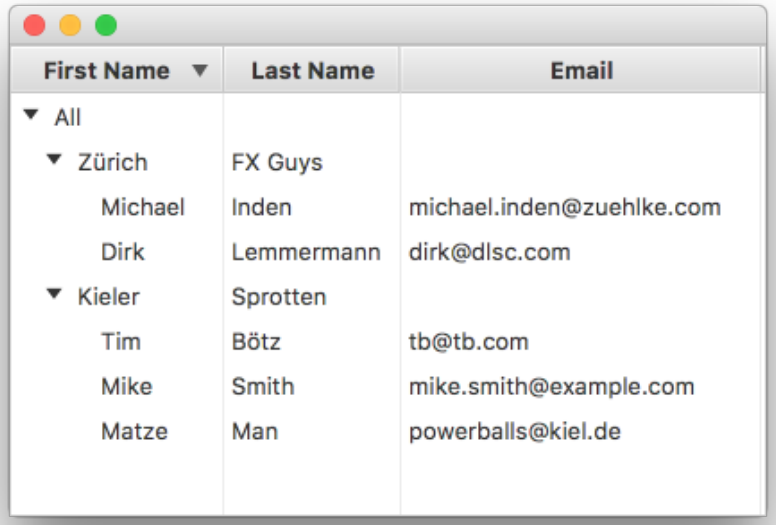
```
birthdayCol.setCellFactory(col -> new LocalDateCell());
birthdayCol.setOnEditCommit(event -> {
    Person localDate = event.getRowValue();
    localDate.setBirthday(event.getNewValue());
});
```



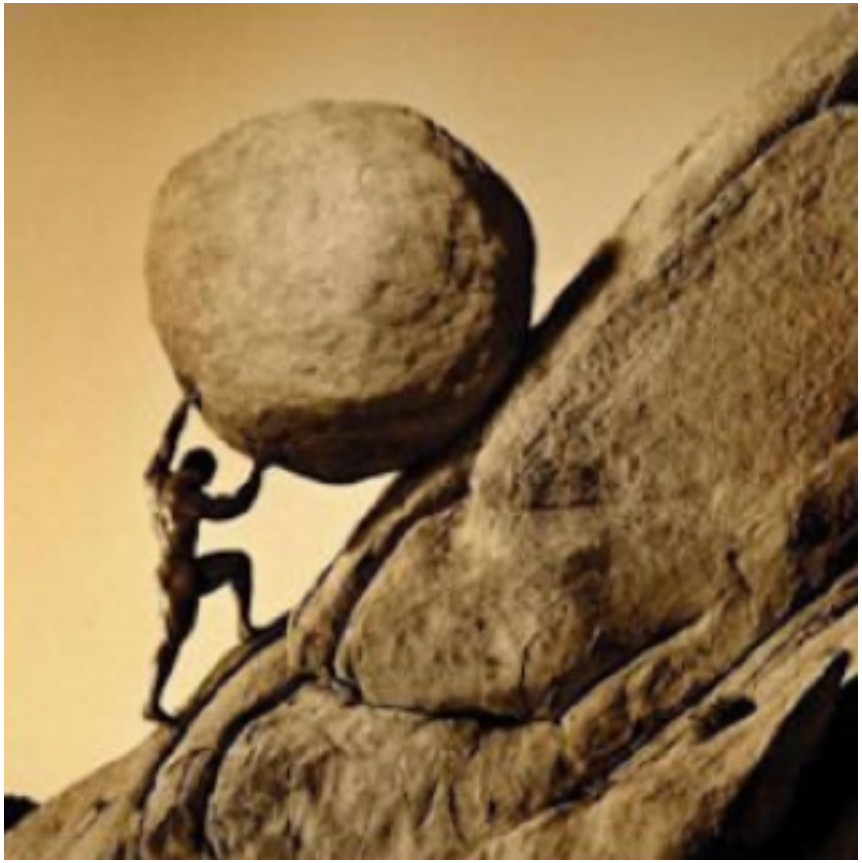
TreeTable – Unterschiede auf einen Blick

zühlke
empowering ideas

```
final TreeItem<Person> root = new TreeItem<>(new Person("All", "", ""));

final TreeTableColumn<Person, String> firstNameCol =
                                        new TreeTableColumn<>("First Name");

firstNameCol.setCellValueFactory(
                new TreeItemPropertyValueFactory<Person, String>("firstName"));

final TreeTableView<Person> treeTableView = new TreeTableView<>(root);
treeTableView.getColumns().addAll(firstNameCol, ... );
```



Übungen Teil 3



zühlke
empowering ideas

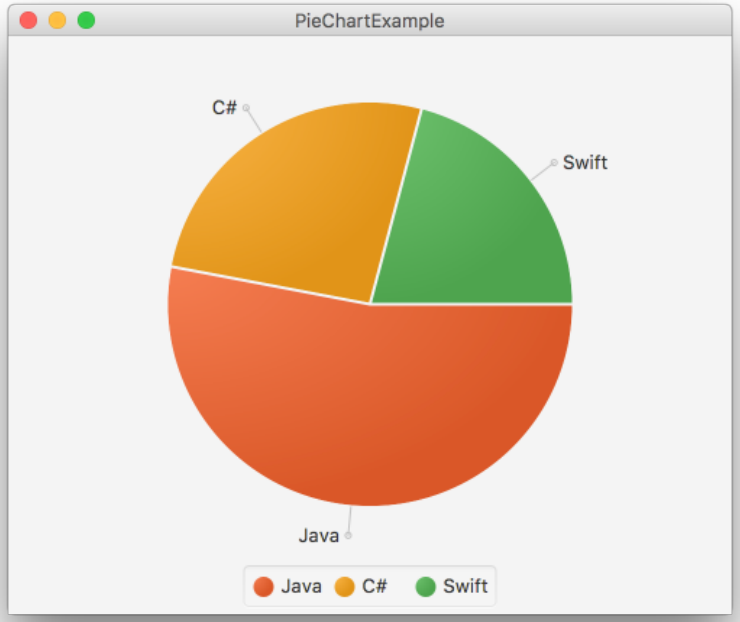
Part 4: Charts

Michael Inden

PieCharts

zühlke
empowering ideas

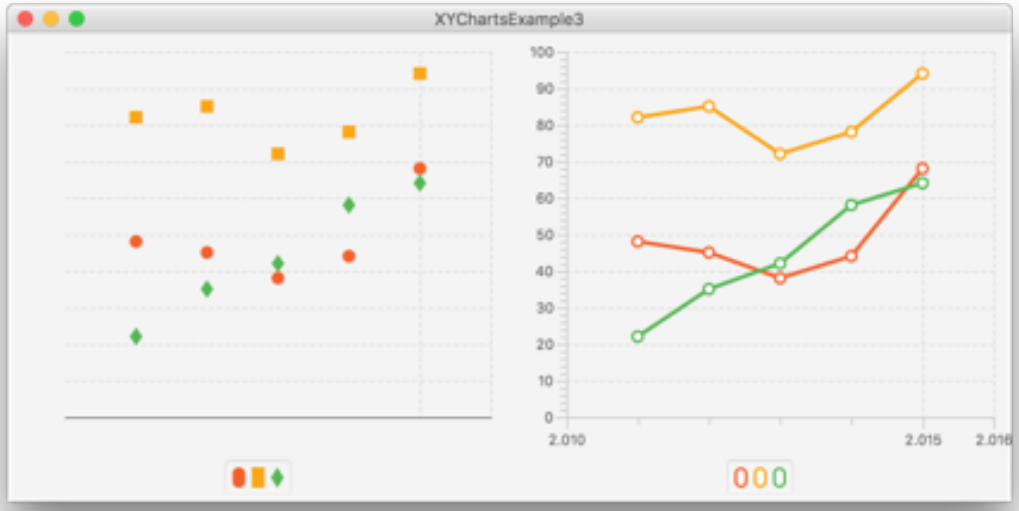
```
@Override
public void start(Stage primaryStage) throws Exception
{
    final PieChart pieChart = new PieChart();
    pieChart.setData(createChartData());

    final StackPane root = new StackPane();
    root.getChildren().add(pieChart);

    primaryStage.setScene(new Scene(root));
    primaryStage.setTitle("PieChartExample");
    primaryStage.show();
}

private ObservableList<Data> createChartData() {
    final ObservableList<Data> data =
       FXCollections.observableArrayList();
    data.add(new Data("Java", 87.65));
    data.add(new Data("C#", 43.21));
    data.add(new Data("Swift", 34.56));
    return data;
}
```



Scatter & LineChart

zühlke
empowering ideas

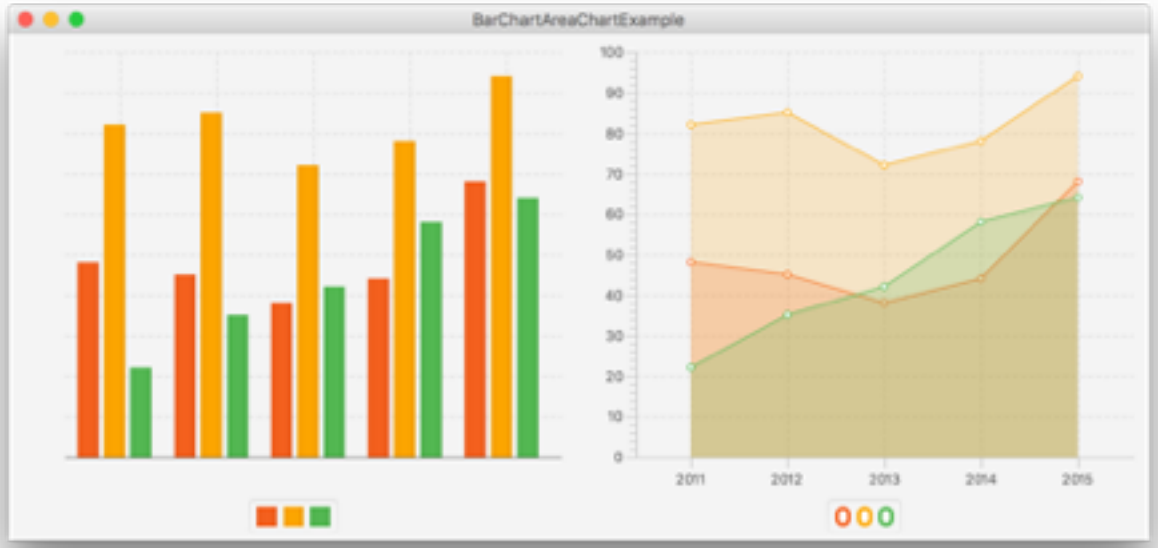
```
final NumberAxis xAxis = new NumberAxis();
xAxis.setAutoRanging(false);
xAxis.setLowerBound(2010);
xAxis.setUpperBound(2016);
final NumberAxis yAxis = new NumberAxis();

final ScatterChart scatterChart = new ScatterChart(xAxis, yAxis);
scatterChart.setData(createChartData());

final LineChart lineChart = new LineChart(xAxis, yAxis);
lineChart.setData(createChartData());
```



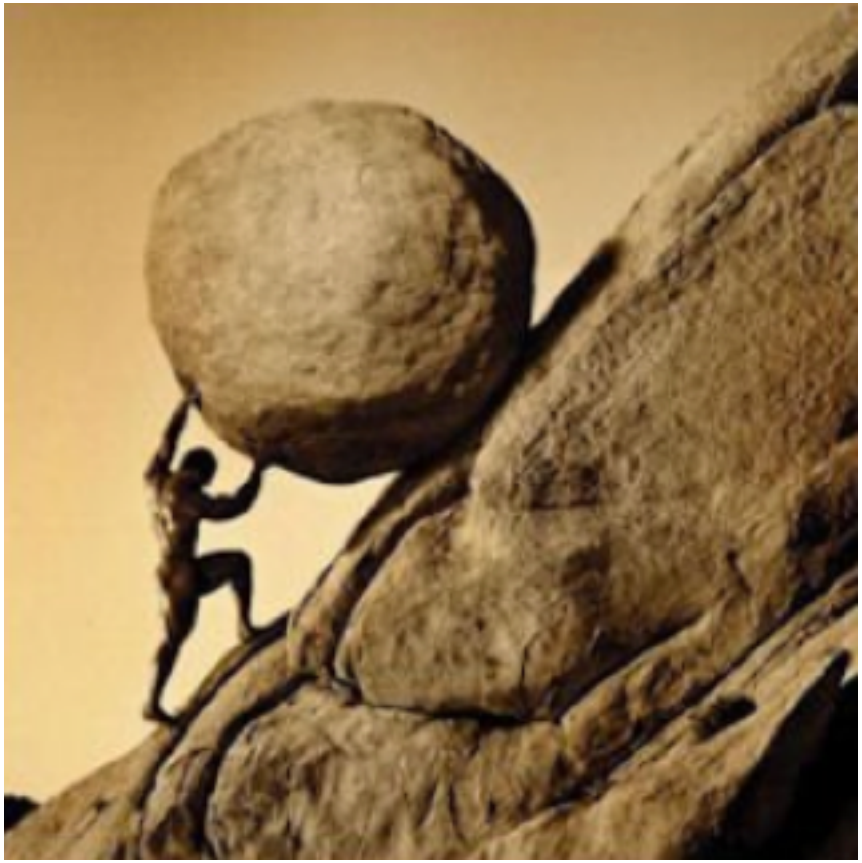
BarChart & AreaChart

zühlke
empowering ideas

```
final CategoryAxis xCategoryAxis = new CategoryAxis();
final NumberAxis yAxis = new NumberAxis();

final BarChart barChart = new BarChart(xCategoryAxis, yAxis);
barChart.setData(createChartData());

final AreaChart areaChart = new AreaChart(xCategoryAxis, yAxis);
areaChart.setData(createChartData());
```



Übungen Teil 4



zühlke
empowering ideas

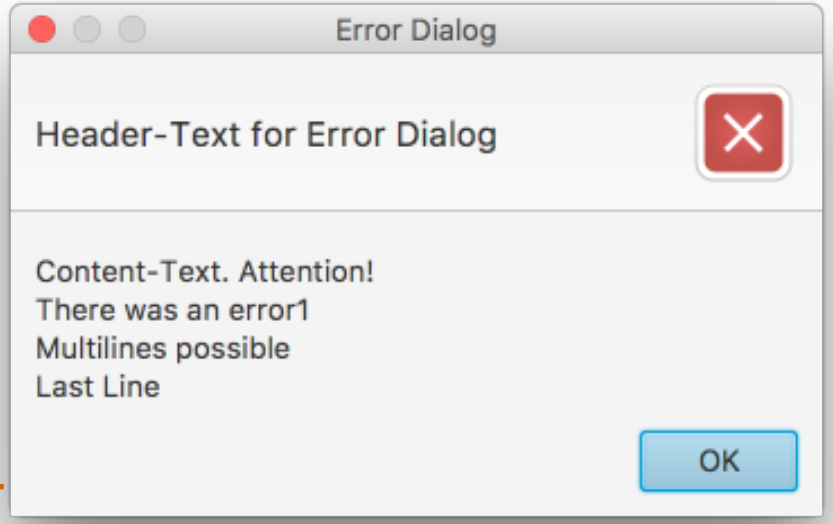
Part 5: Neuerungen in Java 8 Update 40

Michael Inden

Info/Warn/Error-Dialoge

zühlke
empowering ideas

```
@Override
public void start(Stage primaryStage) throws Exception
{
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Error Dialog");
    alert.setHeaderText("Header-Text for Error Dialog");
    alert.setContentText("Content-Text. Attention!\n" +
                         "There was an error1\n" +
                         "Multilines possible\nLast Line");

    alert.showAndWait();
}
```



Eingabe-/Bestätigungs-Dialoge

zühlke
empowering ideas

- **Liefern `Optional` als Rückgabe**
- `TextInputDialog dialog = new TextInputDialog(" initValue");`
- `ChoiceDialog<String> dialog = new ChoiceDialog<>(“initValue", choices);`

```
final Optional<String> result = dialog.showAndWait();
if (result.isPresent())
{
    System.out.println("Wahl: " + result.get());
}
```

- **Java 8-like mit Lambda:**
  `result.ifPresent(meal -> System.out.println("Wahl: " + meal));`

Eingabe-/Bestätigungs-Dialoge

zühlke
empowering ideas

```
@Override
public void start(Stage primaryStage) throws Exception
{
    List<String> choices = Arrays.asList("Pizza", "Steak", ...);

    ChoiceDialog<String> dialog = new ChoiceDialog<>("Steak",
                                                     choices);
    dialog.setTitle("Choice Dialog");
    dialog.setContentText("Wähle dein Menu:");

   result.ifPresent(meal ->
       printMenu(meal));
}
```

Spinner

zühlke
empowering ideas

- auf viele Arten konfigurierbar:
  - Schrittweite
  - verschiedene Varianten der Darstellung/Positionierung der Pfeilknöpfe

FormattedTextField

zühlke
empowering ideas

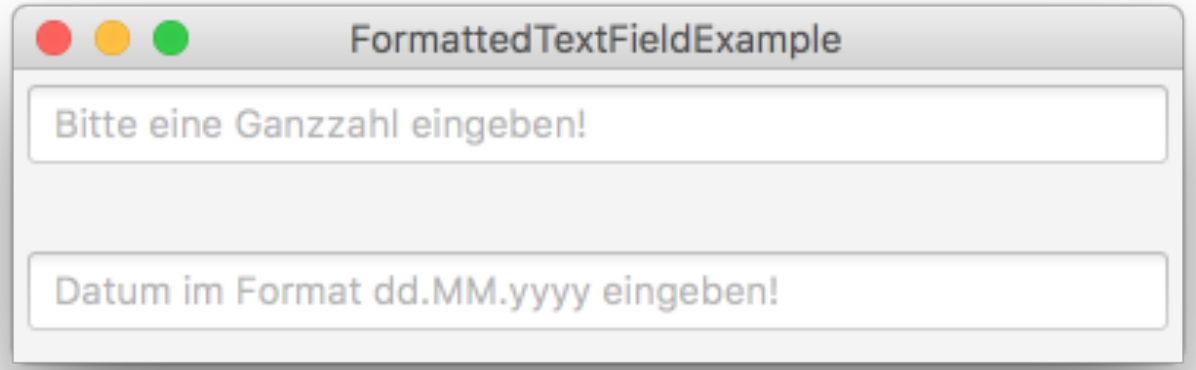
- Erlaubt spezielle Formatierung bzw. Umwandlung der Eingabe
- um Zahlen oder Datumswerte adäquat zu verarbeiten.
- Benötigt Konverterklassen , werden seit Update 40 mitgeliefert, unter anderem:
  IntegerStringConverter, LocalDateStringConverter, TimeStringConverter
- Klasse TextFormatter steuert die Verarbeitung in einem TextField

```
private TextField createFormattedTextField(
                                final StringConverter<?> converter)
{
    final TextField textField = new TextField();

    textField.setTextFormatter(new TextFormatter<>(converter));

    return textField;
}
```

FormattedTextField

zühlke
empowering ideas

```
@Override
public void start(final Stage stage)
{
    StringConverter<Integer> intToString = new IntegerStringConverter();
    TextField integerTextField = createFormattedTextField(intToString);
    integerTextField.setPromptText("Bitte eine Ganzzahl eingeben!");

    StringConverter<LocalDate> dateToString = createLocalDateConverter();
    TextField dateTextField = createFormattedTextField(dateToString);
    dateTextField.setPromptText("Datum im Format dd.MM.yyyy eingeben!");

    ...
}
```



Eigene Converter

zühlke
empowering ideas

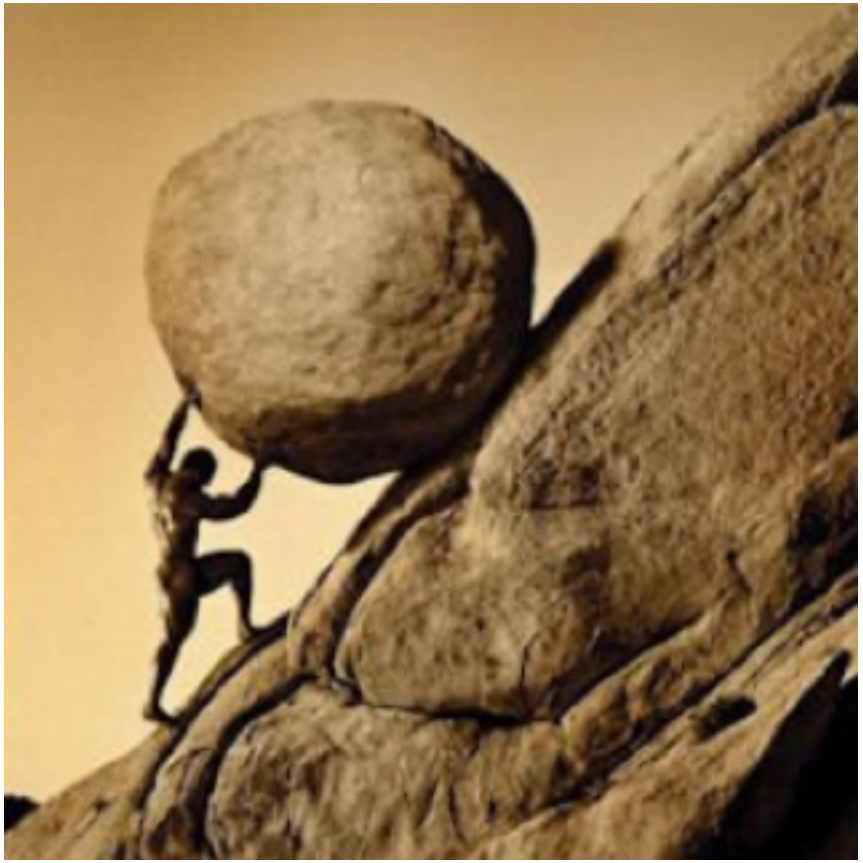
```
private StringConverter<LocalDate> createLocalDateConverter()
{
    return new LocalDateStringConverter(FormatStyle.MEDIUM,
                                        Locale.GERMANY,
                    Chronology.ofLocale(Locale.GERMANY));
}
```

- **Gültigkeitsprüfung nach Bestätigung:**

```
textField.setOnAction(event -> checkValidity(textField,
                                              converter));
```

Übungen Teil 5 + Bonus

zühlke
empowering ideas



zühlke
empowering ideas

Zusammenfassung und Links

Michael Inden

Zusammenfassung

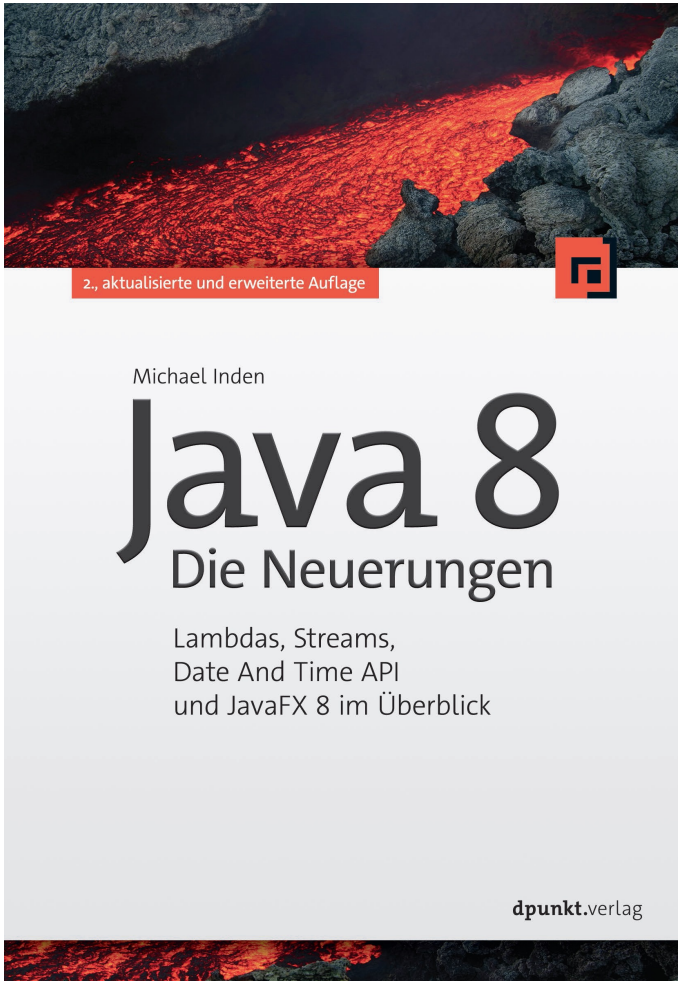
zühlke
empowering ideas

- **JavaFX 8** verschiedene **neue Controls** und Unterstützung für **3D**
- diversen **API-Erweiterungen** eine **Erleichterung** beim täglichen Entwickeln

- Hier nicht behandelt:
  - umfangreiche Effekte und Animationen
  - Zeichnen im Canvas
  - WebView
  - Audio- und Video-Support
  - Migration: Swing in JavaFX (SwingNode), JavaFX in Swing (JFXPanel)

Weiterführende Infos und Links

zühlke
empowering ideas



Weiterführende Infos und Links

zühlke
empowering ideas

JavaFX
http://docs.oracle.com/javafx/

JavaFX Dzone RefCard
https://dzone.com/refcardz/javafx-8-1

JavaFX for Swing Developers
http://docs.oracle.com/javafx/2/swing/jfxpub-swing.htm

SceneBuilder
http://gluonhq.com/open-source/scene-builder/

Getting Started with JavaFX 3D Graphics
http://docs.oracle.com/javafx/8/3d_graphics/jfxpub-3d_graphics.htm

JavaFX 8 Container-Terminal
http://www.youtube.com/embed/AS26gZrYNy8?rel=0

The End

zühlke
empowering ideas

Vielen Dank für die Aufmerksamkeit!

Viel Spaß bei der eigenen Entdeckungsreise zu Java FX 8!