Kostenloses eBook

LERNEN

javafx

Free unaffiliated eBook created from
Stack Overflow contributors.

#javafx

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: javafx

It is an unofficial and free javafx ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official javafx.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Javafx

Bemerkungen

JavaFX ist eine Softwareplattform zum Erstellen und Bereitstellen von Desktopanwendungen sowie Rich-Internet-Anwendungen (RIAs), die auf einer Vielzahl von Geräten ausgeführt werden können. JavaFX soll Swing als Standard-GUI-Bibliothek für Java SE ersetzen.

Mithilfe der IT können Entwickler Rich-Client-Anwendungen entwerfen, erstellen, testen, debuggen und bereitstellen.

Das Erscheinungsbild von JavaFX-Anwendungen kann mithilfe von Cascading Style Sheets (CSS) für das Styling (siehe JavaFX: CSS) angepasst werden. XML-Dateien (F) können für Objektstrukturen verwendet werden, die das Erstellen oder Entwickeln einer Anwendung erleichtern (siehe FXML und Controller) . Scene Builder ist ein visueller Editor, mit dem Sie fxml-Dateien für eine Benutzeroberfläche erstellen können, ohne Code schreiben zu müssen.

Versionen

Ausführung	Veröffentlichungsdatum
JavaFX 2	2011-10-10
JavaFX 8	2014-03-18

Examples

Installation oder Setup

> Die JavaFX-APIs sind als vollständig integrierte Funktion der Java SE Runtime Environment (JRE) und des Java Development Kit (JDK) verfügbar. Da das JDK für alle gängigen Desktop-Plattformen (Windows, Mac OS X und Linux) verfügbar ist, können JavaFX-Anwendungen ab JDK 7 und höher auch auf allen wichtigen Desktop-Plattformen kompiliert werden. Unterstützung für ARM-Plattformen wurde auch mit JavaFX 8 bereitgestellt. JDK for ARM umfasst die Basis-, Grafik- und Steuerelementkomponenten von JavaFX.

Um JavaFX zu installieren, installieren Sie Ihre ausgewählte Version der Java Runtime-Umgebung und des Java Development Kit .

Zu den von JavaFX angebotenen Funktionen gehören:

1. Java-APIs.
2. FXML und Scene Builder.
3. WebView.

4. Swing-Interoperabilität
5. Eingebaute UI-Steuerelemente und CSS.
6. Modena-Thema.
7. 3D-Grafikfunktionen.
8. Canvas-API.
9. API drucken.
10. Rich Text-Unterstützung.
11. Multitouch-Unterstützung.
12. Hi-DPI-Unterstützung.
13. Hardware-beschleunigte Grafikpipeline.
14. Hochleistungs-Media-Engine.
15. Eigenständiges Anwendungsbereitstellungsmodell.

Hallo Weltprogramm

Mit dem folgenden Code wird eine einfache Benutzeroberfläche erstellt, die einen einzelnen `Button`, der beim Klicken einen `String` an die Konsole druckt.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        // create a button with specified text
        Button button = new Button("Say 'Hello World'");

        // set a handler that is executed when the user activates the button
        // e.g. by clicking it or pressing enter while it's focused
        button.setOnAction(e -> {
           //Open information dialog that says hello
           Alert alert = new Alert(AlertType.INFORMATION, "Hello World!?");
           alert.showAndWait();
        });

        // the root of the scene shown in the main window
        StackPane root = new StackPane();

        // add button as child of the root
        root.getChildren().add(button);

        // create a scene specifying the root and the size
        Scene scene = new Scene(root, 500, 300);

        // add scene to the stage
        primaryStage.setScene(scene);

        // make the stage visible
        primaryStage.show();
```

```
    }

    public static void main(String[] args) {
        // launch the HelloWorld application.

        // Since this method is a member of the HelloWorld class the first
        // parameter is not required
        Application.launch(HelloWorld.class, args);
    }

}
```

Die `Application` Klasse ist der Einstiegspunkt jeder JavaFX-Anwendung. Es kann nur eine `Application` gestartet werden. Dies erfolgt mit

```
Application.launch(HelloWorld.class, args);
```

Dadurch wird eine als Parameter übergebene Instanz der `Application` Klasse erstellt und die JavaFX-Plattform gestartet.

Folgendes ist für den Programmierer hier wichtig:

1. Beim ersten `launch` wird eine neue Instanz der `Application` Klasse (in diesem Fall `HelloWorld`) erstellt. Die `Application` Klasse benötigt daher einen No-Arg-Konstruktor.
2. `init()` wird für die erstellte `Application` aufgerufen. In diesem Fall führt die Standardimplementierung von `Application` nichts aus.
3. `start` wird für die `Appication` Instanz `Appication` und die primäre `Stage` (= window) wird an die Methode übergeben. Diese Methode wird automatisch im JavaFX-Anwendungsthread (Plattformthread) aufgerufen.
4. Die Anwendung wird ausgeführt, bis die Plattform feststellt, dass es Zeit ist, herunterzufahren. Dies geschieht, wenn das letzte Fenster in diesem Fall geschlossen wird.
5. Die `stop` Methode wird in der `Application` aufgerufen. In diesem Fall führt die Implementierung von `Application` nichts aus. Diese Methode wird automatisch im JavaFX-Anwendungsthread (Plattformthread) aufgerufen.

Bei der `start` wird der Szenengraph erstellt. In diesem Fall enthält es 2 `Node` s: Ein `Button` und ein `StackPane` .

Die `Button` stellt eine Schaltfläche in der Benutzeroberfläche und der `StackPane` ist ein Container für die `Button` , die es Platzierung bestimmt.

Eine `Scene` wird erstellt, um diese `Node` anzuzeigen. Schließlich wird die `Scene` der `Stage` hinzugefügt, in der die gesamte Benutzeroberfläche angezeigt wird.

Erste Schritte mit Javafx online lesen: https://riptutorial.com/de/javafx/topic/887/erste-schritte-mit-javafx

Kapitel 2: Animation

Examples

Animieren einer Eigenschaft mit der Timeline

```
Button button = new Button("I'm here...");

Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80))
);
t.setAutoReverse(true);
t.setCycleCount(Timeline.INDEFINITE);
t.play();
```

Die grundlegendste und flexibelste Möglichkeit, Animationen in JavaFX zu verwenden, ist die `Timeline` Klasse. Eine Zeitleiste funktioniert, indem `KeyFrame` als bekannte Punkte in der Animation verwendet werden. In diesem Fall weiß er, dass `translateXProperty` am Start (`0 seconds`) Null sein muss und am Ende (`2 seconds`) die Eigenschaft `80` . Sie können auch andere Aktionen ausführen, z. B. die Animation umkehren und wie oft sie ausgeführt werden soll.

Zeitleisten können mehrere Eigenschaften gleichzeitig animieren:

```
Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(1), new KeyValue(button.translateYProperty(), 10)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80)),
        new KeyFrame(Duration.seconds(3), new KeyValue(button.translateYProperty(), 90))
);                                                              //   ^ notice X vs Y
```

Bei dieser Animation wird die `Y` Eigenschaft von `0` (Startwert der Eigenschaft) bis zu einer Sekunde Sekunde auf `10` , und sie endet bei `90` Sekunden bei drei Sekunden. Wenn die Animation erneut beginnt, geht `Y` auf Null zurück, obwohl es nicht der erste Wert in der Timeline ist.

Animation online lesen: https://riptutorial.com/de/javafx/topic/5166/animation

Kapitel 3: CSS

Syntax

- NodeClass / * Selektor nach der Klasse des Knotens * /
- .someclass / * Selektor nach Klasse * /
- #someId / * Selektor nach ID * /
- [Selector1]> [Selector2] / * Selector für ein direktes Kind eines Knotens, das mit Selector1 übereinstimmt, der mit Selector2 übereinstimmt * / /
- [Selector1] [Selector2] / * Selector für einen Nachkomme eines Knotens, der mit Selector1 übereinstimmt und mit Selector2 übereinstimmt * / /

Examples

CSS zum Stylen verwenden

CSS kann an mehreren Stellen angewendet werden:

- inline (`Node.setStyle`)
- in einem Stylesheet
  - zu einer `Scene`
    - als User Agent Stylesheet (hier nicht gezeigt)
    - als "normales" Stylesheet für die `Scene`
  - zu einem `Node`

Dies ermöglicht das Ändern der ansprechbaren Eigenschaften von `Nodes` . Das folgende Beispiel veranschaulicht dies:

Anwendungsklasse

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Region;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class StyledApplication extends Application {

    @Override
    public void start(Stage primaryStage) {

        Region region1 = new Region();
        Region region2 = new Region();
        Region region3 = new Region();
        Region region4 = new Region();
        Region region5 = new Region();
        Region region6 = new Region();
```

```
        // inline style
        region1.setStyle("-fx-background-color: yellow;");

        // set id for styling
        region2.setId("region2");

        // add class for styling
        region2.getStyleClass().add("round");
        region3.getStyleClass().add("round");

        HBox hBox = new HBox(region3, region4, region5);

        VBox vBox = new VBox(region1, hBox, region2, region6);

        Scene scene = new Scene(vBox, 500, 500);

        // add stylesheet for root
        scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());

        // add stylesheet for hBox
        hBox.getStylesheets().add(getClass().getResource("inlinestyle.css").toExternalForm());

        scene.setFill(Color.BLACK);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

inlinestyle.css

```
* {
    -fx-opacity: 0.5;
}

HBox {
    -fx-spacing: 10;
}

Region {
    -fx-background-color: white;
}
```

style.css

```
Region {
    width: 50;
    height: 70;

    -fx-min-width: width;
    -fx-max-width: width;

    -fx-min-height: height;
```

```
    -fx-max-height: height;

    -fx-background-color: red;
}

VBox {
    -fx-spacing: 30;
    -fx-padding: 20;
}

#region2 {
    -fx-background-color: blue;
}
```

Rechteck erweitern, um neue anpassbare Eigenschaften hinzuzufügen

JavaFX 8

Das folgende Beispiel zeigt, wie Sie benutzerdefinierte Eigenschaften, die von CSS zu einem benutzerdefinierten `Node` gestaltet werden können, hinzufügen.

Hier werden 2 `DoubleProperty` zur `Rectangle` Klasse hinzugefügt, um die Einstellung der `width` und `height` über CSS zu ermöglichen.

Das folgende CSS kann zum Gestalten des benutzerdefinierten Knotens verwendet werden:

```
 StyleableRectangle {
    -fx-fill: brown;
    -fx-width: 20;
    -fx-height: 25;
    -fx-cursor: hand;
}
```

Benutzerdefinierter Knoten

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import javafx.beans.property.DoubleProperty;
import javafx.css.CssMetaData;
import javafx.css.SimpleStyleableDoubleProperty;
import javafx.css.StyleConverter;
import javafx.css.Styleable;
import javafx.css.StyleableDoubleProperty;
import javafx.css.StyleableProperty;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Rectangle;

public class StyleableRectangle extends Rectangle {

    // declaration of the new properties
    private final StyleableDoubleProperty styleableWidth = new
SimpleStyleableDoubleProperty(WIDTH_META_DATA, this, "styleableWidth");
    private final StyleableDoubleProperty styleableHeight = new
```

```
SimpleStyleableDoubleProperty(HEIGHT_META_DATA, this, "styleableHeight");

    public StyleableRectangle() {
        bind();
    }

    public StyleableRectangle(double width, double height) {
        super(width, height);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double width, double height, Paint fill) {
        super(width, height, fill);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double x, double y, double width, double height) {
        super(x, y, width, height);
        initStyleableSize();
        bind();
    }

    private void initStyleableSize() {
        styleableWidth.set(getWidth());
        styleableHeight.set(getHeight());
    }

    private final static List<CssMetaData<? extends Styleable, ?>> CLASS_CSS_META_DATA;

    // css metadata for the width property
    // specify property name as -fx-width and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> WIDTH_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-width", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            // property can be set iff the property is not bound
            return !styleable.styleableWidth.isBound();
        }

        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
            // extract the property from the styleable
            return styleable.styleableWidth;
        }
    };

    // css metadata for the height property
    // specify property name as -fx-height and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> HEIGHT_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-height", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            return !styleable.styleableHeight.isBound();
        }
```

```
        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
            return styleable.styleableHeight;
        }
    };

    static {
        // combine already available properties in Rectangle with new properties
        List<CssMetaData<? extends Styleable, ?>> parent = Rectangle.getClassCssMetaData();
        List<CssMetaData<? extends Styleable, ?>> additional = Arrays.asList(HEIGHT_META_DATA,
WIDTH_META_DATA);

        // create arraylist with suitable capacity
        List<CssMetaData<? extends Styleable, ?>> own = new ArrayList(parent.size()+
additional.size());

        // fill list with old and new metadata
        own.addAll(parent);
        own.addAll(additional);

        // make sure the metadata list is not modifiable
        CLASS_CSS_META_DATA = Collections.unmodifiableList(own);
    }

    // make metadata available for extending the class
    public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    // returns a list of the css metadata for the stylable properties of the Node
    @Override
    public List<CssMetaData<? extends Styleable, ?>> getCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    private void bind() {
        this.widthProperty().bind(this.styleableWidth);
        this.heightProperty().bind(this.styleableHeight);
    }


    // ------------------------------------------------------------------------
    // ---------------------- PROPERTY METHODS --------------------------------
    // ------------------------------------------------------------------------

    public final double getStyleableHeight() {
        return this.styleableHeight.get();
    }

    public final void setStyleableHeight(double value) {
        this.styleableHeight.set(value);
    }

    public final DoubleProperty styleableHeightProperty() {
        return this.styleableHeight;
    }

    public final double getStyleableWidth() {
        return this.styleableWidth.get();
    }
```

```
    public final void setStyleableWidth(double value) {
        this.styleableWidth.set(value);
    }

    public final DoubleProperty styleableWidthProperty() {
        return this.styleableWidth;
    }

}
```

CSS online lesen: https://riptutorial.com/de/javafx/topic/1581/css

Kapitel 4: Diagramm

Examples

Kuchendiagramm

Die `PieChart` Klasse zeichnet Daten in Form eines Kreises, der in `PieChart` unterteilt ist. Jedes Segment repräsentiert einen Prozentsatz (Teil) für einen bestimmten Wert. Die Kreisdiagrammdaten werden in `PieChart.Data` Objekten eingeschlossen. Jedes `PieChart.Data` Objekt hat zwei Felder: den Namen des `PieChart.Data` und den entsprechenden Wert.

Konstrukteure

Um ein Kreisdiagramm zu erstellen, müssen Sie das Objekt der `PieChart` Klasse `PieChart` . Zwei Konstrukteure stehen uns zur Verfügung. Einer erstellt ein leeres Diagramm, in dem nichts angezeigt wird, es sei denn, die Daten werden mit der `setData` Methode festgelegt:

```
PieChart pieChart = new PieChart(); // Creates an empty pie chart
```

Die zweite erfordert, dass eine `ObservableList` von `PieChart.Data` als Parameter übergeben wird.

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Cats", 50),
            new PieChart.Data("Dogs", 50));
PieChart pieChart(valueList); // Creates a chart with the given data
```

Daten

Die Werte der Kreissegmente müssen nicht notwendigerweise 100 sein, da die Schnittgröße proportional zur Summe aller Werte berechnet wird.

Die Reihenfolge, in der die Dateneinträge der Liste hinzugefügt werden, bestimmt deren Position im Diagramm. Standardmäßig werden sie im Uhrzeigersinn verlegt. Dieses Verhalten kann jedoch umgekehrt werden:

```
pieChart.setClockwise(false);
```

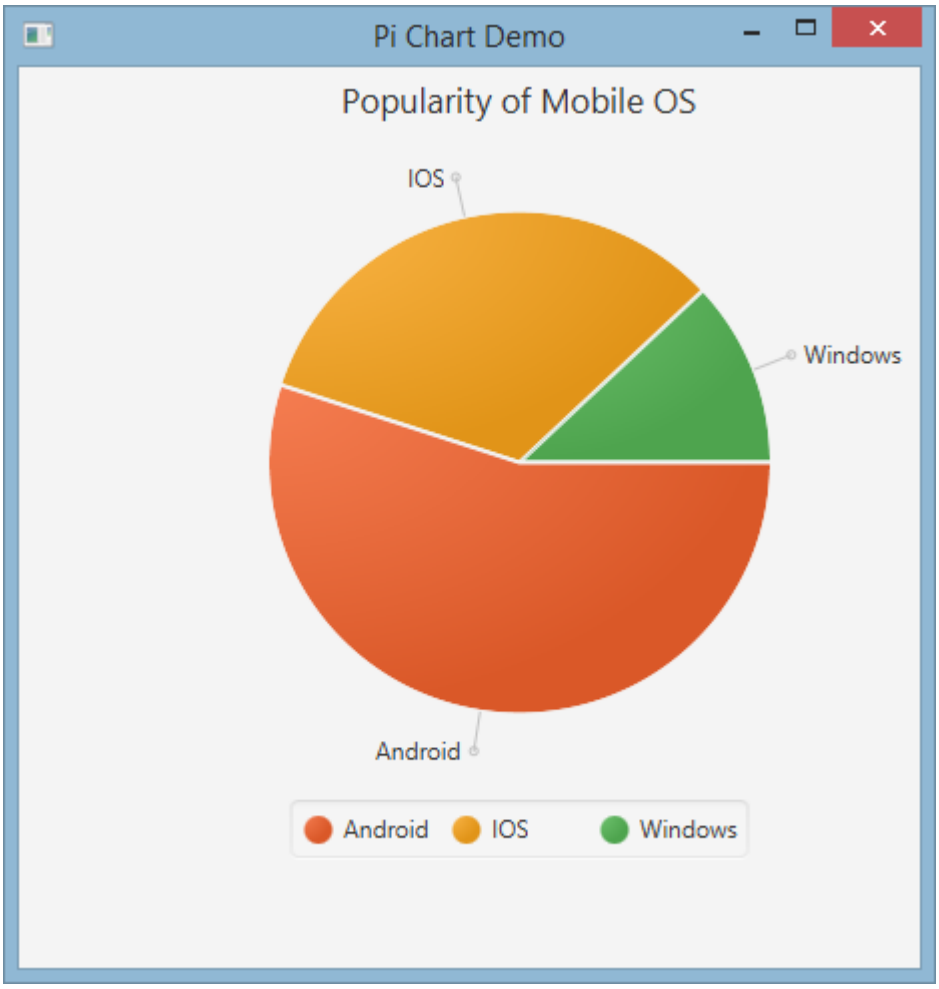
Beispiel

Das folgende Beispiel erstellt ein einfaches Kreisdiagramm:

```
import javafx.application.Application;
```

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class Main extends Application {

@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();
    ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Android", 55),
            new PieChart.Data("IOS", 33),
            new PieChart.Data("Windows", 12));
    // create a pieChart with valueList data.
    PieChart pieChart = new PieChart(valueList);
    pieChart.setTitle("Popularity of Mobile OS");
    //adding pieChart to the root.
    root.getChildren().addAll(pieChart);
    Scene scene = new Scene(root, 450, 450);

    primaryStage.setTitle("Pie Chart Demo");
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

Ausgabe:



Interaktives Kreisdiagramm

Standardmäßig verarbeitet `PieChart` keine Ereignisse. Dieses Verhalten kann jedoch geändert werden, da jeder Kreisabschnitt ein JavaFX- `Node` .

In dem folgenden Beispiel initialisieren wir die Daten, weisen sie dem Diagramm zu und durchlaufen dann den Datensatz, indem jedem Slice QuickInfos hinzugefügt werden, damit die normalerweise verborgenen Werte dem Benutzer angezeigt werden können.

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Nitrogen", 7809),
            new PieChart.Data("Oxygen", 2195),
            new PieChart.Data("Other", 93));

PieChart pieChart = new PieChart(valueList);
pieChart.setTitle("Air composition");

pieChart.getData().forEach(data -> {
    String percentage = String.format("%.2f%%", (data.getPieValue() / 100));
    Tooltip toolTip = new Tooltip(percentage);
    Tooltip.install(data.getNode(), toolTip);
});
```

Liniendiagramm

Die `LineChart` Klasse stellt die Daten als eine Reihe von Datenpunkten dar, die mit geraden Linien verbunden sind. Jeder Datenpunkt wird in ein `XYChart.Data` Objekt eingeschlossen und die Datenpunkte werden in `XYChart.Series` gruppiert.

Jedes `XYChart.Data` Objekt hat zwei Felder, auf die mit `getXValue` und `getYValue` zugegriffen werden `getXValue` , die einem x- und einem y-Wert in einem Diagramm entsprechen.

```
XYChart.Data data = new XYChart.Data(1,3);
System.out.println(data.getXValue()); // Will print 1
System.out.println(data.getYValue()); // Will print 3
```

Äxte

Bevor wir ein `LineChart` erstellen, `LineChart` wir seine Achsen definieren. Der Standardkonstruktor ohne Argumente einer `NumberAxis` Klasse erstellt beispielsweise eine Auto-Ranging-Achse, die `NumberAxis` verwendet werden kann und keine weitere Konfiguration erfordert.

```
Axis xAxis = new NumberAxis();
```

Beispiel

Im folgenden Beispiel erstellen wir zwei Datenreihen, die in demselben Diagramm angezeigt werden. Die Beschriftungen, Bereiche und Tick-Werte der Achsen werden explizit definiert.

```
@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();

    // Create empty series
    ObservableList<XYChart.Series> seriesList = FXCollections.observableArrayList();

    // Create data set for the first employee and add it to the series
    ObservableList<XYChart.Data> aList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 6),
            new XYChart.Data(4, 37),
            new XYChart.Data(6, 82),
            new XYChart.Data(8, 115)
    );
    seriesList.add(new XYChart.Series("Employee A", aList));

    // Create data set for the second employee and add it to the series
    ObservableList<XYChart.Data> bList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 43),
            new XYChart.Data(4, 51),
            new XYChart.Data(6, 64),
            new XYChart.Data(8, 92)
    );
    seriesList.add(new XYChart.Series("Employee B", bList));
```

```
    // Create axes
    Axis xAxis = new NumberAxis("Hours worked", 0, 8, 1);
    Axis yAxis = new NumberAxis("Lines written", 0, 150, 10);

    LineChart chart = new LineChart(xAxis, yAxis, seriesList);

    root.getChildren().add(chart);

    Scene scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
}

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

Ausgabe:

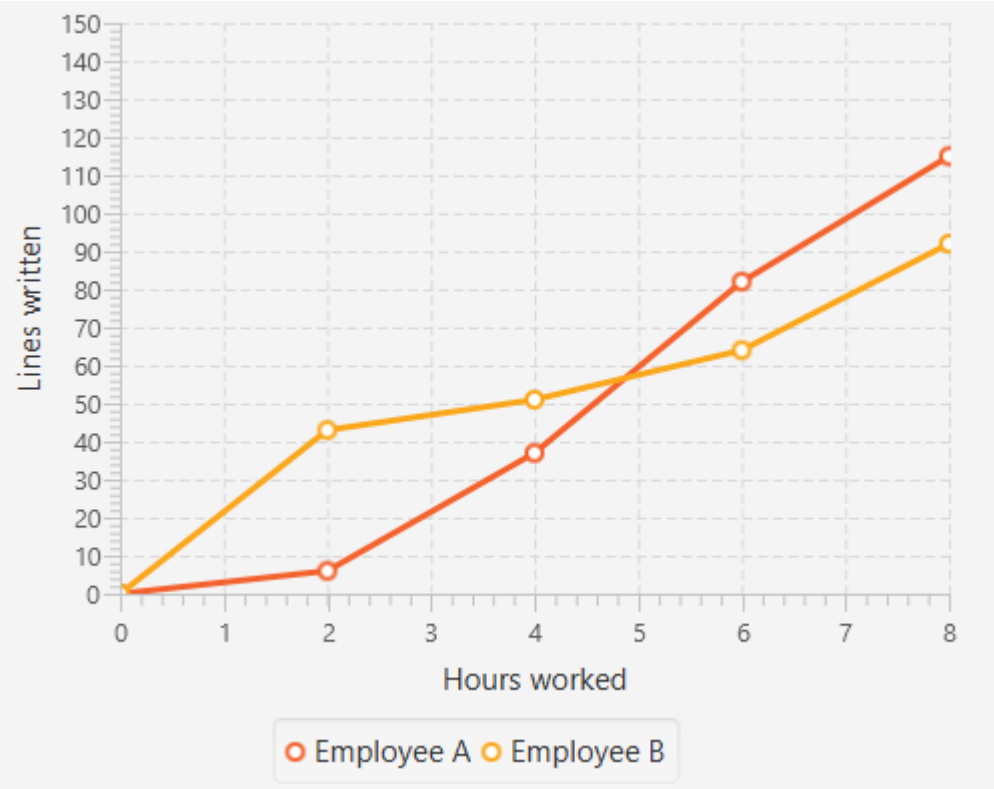


Diagramm online lesen: https://riptutorial.com/de/javafx/topic/2631/diagramm

Kapitel 5: Dialoge

Bemerkungen

In JavaFX 8 Update 40 wurden Dialoge hinzugefügt.

Examples

TextInputDialog

`TextInputDialog` ermöglicht die den Benutzer zur Eingabe ein einzelnes fragen `String` .

```
TextInputDialog dialog = new TextInputDialog("42");
dialog.setHeaderText("Input your favourite int.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite int: ");

Optional<String> result = dialog.showAndWait();

String s = result.map(r -> {
    try {
        Integer n = Integer.valueOf(r);
        return MessageFormat.format("Nice! I like {0} too!", n);
    } catch (NumberFormatException ex) {
        return MessageFormat.format("Unfortunately \"{0}\" is not a int!", r);
    }
}).orElse("You really don't want to tell me, huh?");

System.out.println(s);
```

ChoiceDialog

`ChoiceDialog` ermöglicht es dem Benutzer, ein Element aus einer Liste von Optionen auszuwählen.

```
List<String> options = new ArrayList<>();
options.add("42");
options.add("9");
options.add("467829");
options.add("Other");

ChoiceDialog<String> dialog = new ChoiceDialog<>(options.get(0), options);
dialog.setHeaderText("Choose your favourite number.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite number:");

Optional<String> choice = dialog.showAndWait();

String s = choice.map(c -> "Other".equals(c) ?
              "Unfortunately your favourite number is not available!"
              : "Nice! I like " + c + " too!")
           .orElse("You really don't want to tell me, huh?");
```

```
System.out.println(s);
```

Warnen

`Alert` ist ein einfaches Popup, das eine Reihe von Schaltflächen anzeigt und je nach der Schaltfläche, auf die der Benutzer geklickt hat, ein Ergebnis angezeigt wird:

Beispiel

Dadurch kann der Benutzer entscheiden, ob er die Hauptstufe wirklich beenden möchte:

```
@Override
public void start(Stage primaryStage) {
    Scene scene = new Scene(new Group(), 100, 100);

    primaryStage.setOnCloseRequest(evt -> {
        // allow user to decide between yes and no
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Do you really want to close
this application?", ButtonType.YES, ButtonType.NO);

        // clicking X also means no
        ButtonType result = alert.showAndWait().orElse(ButtonType.NO);

        if (ButtonType.NO.equals(result)) {
            // consume event i.e. ignore close request
            evt.consume();
        }
    });
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Beachten Sie, dass der Schaltflächentext automatisch an das `Locale` angepasst wird.

Text für benutzerdefinierte Schaltflächen

Der in einer Schaltfläche angezeigte Text kann angepasst werden, indem `ButtonType` selbst eine `ButtonType` Instanz `ButtonType` :

```
ButtonType answer = new ButtonType("42");
ButtonType somethingElse = new ButtonType("54");

Alert alert = new Alert(Alert.AlertType.NONE, "What do you get when you multiply six by
nine?", answer, somethingElse);
ButtonType result = alert.showAndWait().orElse(somethingElse);

Alert resultDialog = new Alert(Alert.AlertType.INFORMATION,
                               answer.equals(result) ? "Correct" : "wrong",
                               ButtonType.OK);

resultDialog.show();
```

Dialoge online lesen: https://riptutorial.com/de/javafx/topic/3681/dialoge

Kapitel 6: Drucken

Examples

Grundlegendes Drucken

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.printPage(some-node);
    if (success) {
        pJ.endJob();
    }
}
```

Dadurch wird auf dem Standarddrucker gedruckt, ohne dass dem Benutzer ein Dialogfeld angezeigt wird. Um einen anderen Drucker als den Standarddrucker zu verwenden, können Sie den `PrinterJob#createPrinterJob(Printer)` zum Einstellen des aktuellen Druckers verwenden. Sie können dies verwenden, um alle Drucker in Ihrem System anzuzeigen:

```
System.out.println(Printer.getAllPrinters());
```

Drucken mit Systemdialog

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.showPrintDialog(primaryStage);// this is the important line
    if (success) {
        pJ.endJob();
    }
}
```

Drucken online lesen: https://riptutorial.com/de/javafx/topic/5157/drucken

Kapitel 7: Eigenschaften & beobachtbar

Bemerkungen

Eigenschaften sind beobachtbar und Listener können hinzugefügt werden. Sie werden konsequent für die Eigenschaften von `Node` .

Examples

Arten von Eigenschaften und Benennung

Standardeigenschaften

Je nach Art der Eigenschaft gibt es bis zu 3 Methoden für eine einzelne Eigenschaft. Lassen Sie `<property>` den Namen einer Eigenschaft und `<Property>` den Namen der Eigenschaft mit einem Großbuchstaben angeben. Und sei `T` der Typ der Immobilie; Für primitive Wrapper verwenden wir hier den primitiven Typ, z. B. `String` für `StringProperty` und `double` für `ReadOnlyDoubleProperty` .

Methodenname	Parameter	Rückgabetyp	Zweck
`<property>Property`	`()`	Die Immobilie selbst, z `DoubleProperty` , `ReadOnlyStringProperty` , `ObjectProperty<VPos>`	Gibt die Eigenschaft selbst zurück, um Listener / Binding hinzuzufügen
`get<Property>`	`()`	`T`	Gibt den in der Eigenschaft umschlossenen Wert zurück
`set<Property>`	`(T)`	`void`	Legen Sie den Wert der Eigenschaft fest

Beachten Sie, dass der Setter für Readonly-Eigenschaften nicht vorhanden ist.

Listet nur Eigenschaften auf

Readonly-Listeneigenschaften sind Eigenschaften, die nur eine Getter-Methode bereitstellen. Der Typ einer solchen Eigenschaft ist `ObservableList` , vorzugsweise mit dem angegebenen Typ agrument. Der Wert dieser Eigenschaft ändert sich nie. Der Inhalt der `ObservableList` kann stattdessen geändert werden.

Readonly-Karteneigenschaften

Ähnlich wie bei Readonly-Listeneigenschaften bieten Readonly-Karteneigenschaften nur einen

Getter, und der Inhalt kann anstelle des Eigenschaftswerts geändert werden. Der Getter gibt eine `ObservableMap` .

StringProperty-Beispiel

Das folgende Beispiel zeigt die Deklaration einer Eigenschaft (in diesem Fall `StringProperty`) und zeigt, wie ein `ChangeListener` wird.

```
import java.text.MessageFormat;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Person {

    private final StringProperty name = new SimpleStringProperty();

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.nameProperty().addListener(new ChangeListener<String>() {

            @Override
            public void changed(ObservableValue<? extends String> observable, String oldValue,
String newValue) {
                System.out.println(MessageFormat.format("The name changed from \"{0}\" to
\"{1}\"", oldValue, newValue));
            }

        });

        person.setName("Anakin Skywalker");
        person.setName("Darth Vader");
    }

}
```

ReadOnlyIntegerProperty-Beispiel

In diesem Beispiel wird gezeigt, wie Sie mit einer readonly-Wrapper-Eigenschaft eine Eigenschaft erstellen, in die nicht geschrieben werden kann. In diesem Fall können die `cost` und der `price` geändert werden, der `profit` jedoch immer der `price - cost` .

```
import java.text.MessageFormat;
```

```
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ReadOnlyIntegerProperty;
import javafx.beans.property.ReadOnlyIntegerWrapper;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Product {

    private final IntegerProperty price = new SimpleIntegerProperty();
    private final IntegerProperty cost = new SimpleIntegerProperty();
    private final ReadOnlyIntegerWrapper profit = new ReadOnlyIntegerWrapper();

    public Product() {
        // the property itself can be written to
        profit.bind(price.subtract(cost));
    }

    public final int getCost() {
        return this.cost.get();
    }

    public final void setCost(int value) {
        this.cost.set(value);
    }

    public final IntegerProperty costProperty() {
        return this.cost;
    }

    public final int getPrice() {
        return this.price.get();
    }

    public final void setPrice(int value) {
        this.price.set(value);
    }

    public final IntegerProperty priceProperty() {
        return this.price;
    }

    public final int getProfit() {
        return this.profit.get();
    }

    public final ReadOnlyIntegerProperty profitProperty() {
        // return a readonly view of the property
        return this.profit.getReadOnlyProperty();
    }

    public static void main(String[] args) {
        Product product = new Product();
        product.profitProperty().addListener(new ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> observable, Number oldValue,
Number newValue) {
                System.out.println(MessageFormat.format("The profit changed from {0}$ to
{1}$", oldValue, newValue));
            }
```

```
        });
        product.setCost(40);
        product.setPrice(50);
        product.setCost(20);
        product.setPrice(30);
    }

}
```

Eigenschaften & beobachtbar online lesen:
https://riptutorial.com/de/javafx/topic/4436/eigenschaften--amp--beobachtbar

Kapitel 8: Einfädeln

Examples

Aktualisieren der Benutzeroberfläche mit Platform.runLater

Langfristige Vorgänge dürfen nicht auf dem JavaFX-Anwendungsthread ausgeführt werden, da JavaFX die Benutzeroberfläche nicht aktualisieren kann, was zu einer eingefrorenen Benutzeroberfläche führt.

Darüber hinaus muss jede Änderung an einem `Node` , der Teil eines "Live" -Szenendiagramms **ist** , im JavaFX-Anwendungsthread vorgenommen werden. `Platform.runLater` kann verwendet werden, um diese Aktualisierungen im JavaFX-Anwendungsthread auszuführen.

Das folgende Beispiel zeigt , wie ein aktualisieren `Text Node` wiederholt von einem anderen Thread:

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CounterApp extends Application {

    private int count = 0;
    private final Text text = new Text(Integer.toString(count));

    private void incrementCount() {
        count++;
        text.setText(Integer.toString(count));
    }

    @Override
    public void start(Stage primaryStage) {
        StackPane root = new StackPane();
        root.getChildren().add(text);

        Scene scene = new Scene(root, 200, 200);

        // longrunning operation runs on different thread
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {
                Runnable updater = new Runnable() {

                    @Override
                    public void run() {
                        incrementCount();
                    }
                };

                while (true) {
```

```
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ex) {
                    }

                    // UI update is run on the Application thread
                    Platform.runLater(updater);
                }
            }

        });
        // don't let thread prevent JVM shutdown
        thread.setDaemon(true);
        thread.start();

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Aktualisieren der Benutzeroberfläche

Mit dem folgenden Code reagiert die Benutzeroberfläche nach dem Klicken der Schaltfläche kurze Zeit nicht, da zu viele Aufrufe von `Platform.runLater` verwendet werden. (Scrollen Sie die `ListView` sofort nach dem Klicken der Schaltfläche.)

```
@Override
public void start(Stage primaryStage) {
    ObservableList<Integer> data = FXCollections.observableArrayList();
    ListView<Integer> listView = new ListView<>(data);

    Button btn = new Button("Say 'Hello World'");
    btn.setOnAction((ActionEvent event) -> {
        new Thread(() -> {
            for (int i = 0; i < 100000; i++) {
                final int index = i;
                Platform.runLater(() -> data.add(index));
            }
        }).start();
    });

    Scene scene = new Scene(new VBox(listView, btn));

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Um dies zu verhindern, anstatt eine große Anzahl von Updates zu verwenden, verwendet der folgende Code einen `AnimationTimer` , um das Update nur einmal pro Frame auszuführen:

```
import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.AnimationTimer;

public class Updater {

    @FunctionalInterface
    public static interface UpdateTask {

        public void update() throws Exception;
    }

    private final List<UpdateTask> updates = new ArrayList<>();

    private final AnimationTimer timer = new AnimationTimer() {

        @Override
        public void handle(long now) {
            synchronized (updates) {
                for (UpdateTask r : updates) {
                    try {
                        r.update();
                    } catch (Exception ex) {
                        Logger.getLogger(Updater.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                updates.clear();
                stop();
            }
        }
    };

    public void addTask(UpdateTask... tasks) {
        synchronized (updates) {
            updates.addAll(Arrays.asList(tasks));
            timer.start();
        }
    }

}
```

Dadurch können die Updates mit der `Updater` Klasse gruppiert werden:

```
private final Updater updater = new Updater();

...

        // Platform.runLater(() -> data.add(index));
        updater.addTask(() -> data.add(index));
```

So verwenden Sie den JavaFX-Service

Anstatt intensive Aufgaben in `JavaFX Thread` auszuführen, sollte dies in einem `Service` ausgeführt werden. Was ist im Grunde ein Dienst ?

Ein Service ist eine Klasse, die bei jedem Start einen neuen `Thread` und eine Aufgabe an sie

weitergibt, um Arbeiten auszuführen. Der Service kann einen Wert zurückgeben oder nicht.

Im Folgenden finden Sie ein typisches Beispiel für den JavaFX-Service, der einige Arbeit leistet und `Map<String,String>()` `Map<String,String>(`:

```
public class WorkerService extends Service<Map<String, String>> {

    /**
     * Constructor
     */
    public WorkerService () {

        // if succeeded
        setOnSucceeded(s -> {
            //code if Service succeeds
        });

        // if failed
        setOnFailed(fail -> {
            //code it Service fails
        });

        //if cancelled
        setOnCancelled(cancelled->{
            //code if Service get's cancelled
        });
    }

    /**
    * This method starts the Service
    */
    public void startTheService(){
        if(!isRunning()){
           //...
           reset();
           start();
        }

    }

    @Override
    protected Task<Map<String, String>> createTask() {
        return new Task<Map<String, String>>() {
            @Override
            protected Void call() throws Exception {

                    //create a Map<String, String>
                    Map<String,String> map  = new HashMap<>();

                   //create other variables here

                   try{
                        //some code here
                        //.....do your manipulation here

                        updateProgress(++currentProgress, totalProgress);
                    }

                } catch (Exception ex) {
                    return null; //something bad happened so you have to do something instead
```

```
of returning null
                }

                return map;
            }
        };
    }

}
```

Einfädeln online lesen: https://riptutorial.com/de/javafx/topic/2230/einfadeln

Kapitel 9: FXML und Controller

Syntax

- xmlns: fx = " http://javafx.com/fxml " // Namespace-Deklaration

Examples

Beispiel FXML

Ein einfaches FXML-Dokument, in dem ein `AnchorPane` mit einer Schaltfläche und einem Beschriftungsknoten beschrieben wird:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml/1"
        fx:controller="com.example.FXMLDocumentController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction"
fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

Diese FXML-Beispieldatei ist einer Controller-Klasse zugeordnet. Die Zuordnung zwischen FXML und Controller-Klasse erfolgt in diesem Fall durch Angabe des Klassennamens als Wert des Attributs `fx:controller` im `fx:controller="com.example.FXMLDocumentController"` der FXML: `fx:controller="com.example.FXMLDocumentController"` . Die Controller-Klasse ermöglicht die Ausführung von Java-Code als Reaktion auf Benutzeraktionen für die in der FXML-Datei definierten Benutzeroberflächenelemente:

```
package com.example ;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

public class FXMLDocumentController {

    @FXML
    private Label label;
```

```
    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle resources) {
        // Initialization code can go here.
        // The parameters url and resources can be omitted if they are not needed
    }

}
```

Ein `FXMLLoader` kann zum Laden der FXML-Datei verwendet werden:

```
public class MyApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("FXMLDocument.fxml"));
        Parent root = loader.load();

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

}
```

Die `load` führt mehrere Aktionen aus, und es ist nützlich, die Reihenfolge zu verstehen, in der sie stattfinden. In diesem einfachen Beispiel:

1. Der `FXMLLoader` liest und analysiert die FXML-Datei. Es erstellt Objekte, die den in der Datei definierten Elementen entsprechen, und notiert die für sie definierten `fx:id` Attribute.

2. Da das `FXMLLoader` der FXML-Datei ein `fx:controller` Attribut definiert hat, erstellt der `FXMLLoader` eine *neue Instanz der* von ihm angegebenen Klasse. Standardmäßig geschieht dies durch Aufrufen des Konstruktors ohne Argumente für die angegebene Klasse.

3. Alle Elemente mit `fx:id` Attributen, die über Felder im Controller mit übereinstimmenden Feldnamen verfügen und die entweder `public` (nicht empfohlen) oder mit `@FXML` (empfohlen) gekennzeichnet sind, werden in die entsprechenden Felder " `@FXML` ". In diesem Beispiel gibt es also in der FXML-Datei ein `Label` mit `fx:id="label"` und einem Feld im Controller, das als definiert ist

   ```
   @FXML
   private Label label ;
   ```

   Das `label` Feld wird mit der vom `FXMLLoader` erstellten `Label` Instanz `FXMLLoader` .

4. Event-Handler werden mit beliebigen Elementen in der FXML-Datei registriert, wobei die Eigenschaften `onXXX="#..."` definiert sind. Diese Ereignishandler rufen die angegebene Methode in der Controller-Klasse auf. In diesem Beispiel hat der `Button` `onAction="#handleButtonAction"` und der Controller definiert eine Methode

   ```
   @FXML
   private void handleButtonAction(ActionEvent event) { ... }
   ```

   Wenn eine Aktion auf der Schaltfläche ausgelöst wird (z. B. wenn der Benutzer sie drückt), wird diese Methode aufgerufen. Die Methode muss einen `void` Rückgabetyp haben und kann entweder einen Parameter definieren, der dem Ereignistyp entspricht (in diesem Beispiel `ActionEvent`), oder er kann keine Parameter definieren.

5. Wenn die Controller-Klasse eine `initialize` definiert, wird diese Methode schließlich aufgerufen. Beachten Sie, dass dies geschieht, nachdem die `@FXML` Felder `@FXML` wurden. Sie können also mit dieser Methode sicher auf sie `@FXML` und werden mit den Instanzen initialisiert, die den Elementen in der FXML-Datei entsprechen. Die Methode `initialize()` kann entweder keine Parameter oder eine `URL` und ein `ResourceBundle` . Im letzteren Fall werden diese Parameter mit der `URL` `FXMLLoader` die den Speicherort der FXML-Datei darstellt, und einem beliebigen `ResourceBundle` , das im `FXMLLoader` über `loader.setResources(...)` . Beide können `null` wenn sie nicht festgelegt wurden.

Verschachtelte Controller

Es ist nicht erforderlich, die gesamte Benutzeroberfläche in einer einzigen FXML mit einem einzigen Controller zu erstellen.

Mit dem `<fx:include>` -Tag kann eine fxml-Datei in eine andere eingefügt werden. Der Controller der mitgelieferten fxml kann wie jedes andere Objekt, das vom `FXMLLoader` erstellt wurde, in den Controller der enthaltenen Datei `FXMLLoader` .

Dies geschieht durch Hinzufügen des Attributs `fx:id` zum Element `<fx:include>` . Auf diese Weise wird der Controller der mitgelieferten fxml in das Feld mit dem Namen `<fx:id value>Controller` eingefügt.

Beispiele:

fx: ID-Wert	Feldname für die Injektion
foo	fooController
antwort42	answer42Controller
xYz	xYzController

Beispiel fxmls

Zähler

Dies ist ein fxml ein enthält `StackPane` mit einem `Text` - Knoten. Der Controller für diese fxml-Datei ermöglicht das Abrufen des aktuellen Zählerwerts sowie das Inkrementieren des Zählers:

counter.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<StackPane prefHeight="200" prefWidth="200" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="counter.CounterController">
    <children>
        <Text fx:id="counter" />
    </children>
</StackPane>
```

CounterController

```
package counter;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class CounterController {
    @FXML
    private Text counter;

    private int value = 0;

    public void initialize() {
        counter.setText(Integer.toString(value));
    }

    public void increment() {
        value++;
        counter.setText(Integer.toString(value));
    }

    public int getValue() {
        return value;
    }

}
```

Einschließlich fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane prefHeight="500" prefWidth="500" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="counter.OuterController">
    <left>
        <Button BorderPane.alignment="CENTER" text="increment" onAction="#increment" />
```

```
    </left>
    <center>
        <!-- content from counter.fxml included here -->
        <fx:include fx:id="count" source="counter.fxml" />
    </center>
</BorderPane>
```

OuterController

Der Controller der mitgelieferten fxml wird in diesen Controller injiziert. Hier wird der Handler für das `onAction` Ereignis für den `Button` verwendet, um den Zähler zu `onAction` .

```
package counter;

import javafx.fxml.FXML;

public class OuterController {

    // controller of counter.fxml injected here
    @FXML
    private CounterController countController;

    public void initialize() {
        // controller available in initialize method
        System.out.println("Current value: " + countController.getValue());
    }

    @FXML
    private void increment() {
        countController.increment();
    }

}
```

Die fxmls können wie folgt geladen werden, vorausgesetzt, der Code wird von einer Klasse in demselben Paket wie `outer.fxml` :

```
Parent parent = FXMLLoader.load(getClass().getResource("outer.fxml"));
```

Blöcke definieren und

Manchmal muss ein Element außerhalb der üblichen Objektstruktur in der fxml erstellt werden.

Hier kommen *Define Blocks* ins Spiel:

Inhalte in einem `<fx:define>` -Element werden nicht zu dem Objekt hinzugefügt, das für das übergeordnete Element erstellt wurde.

Jedes untergeordnete Element von `<fx:define>` benötigt ein `fx:id` Attribut.

Auf diese Weise erstellte Objekte können später mit dem Element `<fx:reference>` oder mit der Ausdrucksbindung `<fx:reference>` .

Das Element `<fx:reference>` kann verwendet werden, um auf ein beliebiges Element mit einem

`fx:id` Attribut zu verweisen, das vor dem `<fx:reference>` -Element behandelt wird, indem derselbe Wert verwendet wird wie das `fx:id` Attribut des referenzierten Elements in `source` - Attribut des `<fx:reference>` Element.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>


<VBox xmlns:fx="http://javafx.com/fxml/1" prefHeight="300.0" prefWidth="300.0"
xmlns="http://javafx.com/javafx/8">
    <children>
        <fx:define>
            <String fx:value="My radio group" fx:id="text" />
        </fx:define>
        <Text>
            <text>
                <!-- reference text defined above using fx:reference -->
                <fx:reference source="text"/>
            </text>
        </Text>
        <RadioButton text="Radio 1">
            <toggleGroup>
                <ToggleGroup fx:id="group" />
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 2">
            <toggleGroup>
                <!-- reference ToggleGroup created for last RadioButton -->
                <fx:reference source="group"/>
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 3" toggleGroup="$group" />

        <!-- reference text defined above using expression binding -->
        <Text text="$text" />
    </children>
</VBox>
```

Daten an FXML übergeben - Zugriff auf vorhandene Controller

Problem: Einige Daten müssen an eine Szene übergeben werden, die aus einem Fxml geladen wurde.

Lösung

Geben Sie einen Controller mit dem Attribut `fx:controller` und `FXMLLoader` die während des Ladevorgangs erstellte Controller-Instanz aus der zum Laden des fxml verwendeten `FXMLLoader` Instanz ab.

Fügen Sie Methoden hinzu, um die Daten an die Controller-Instanz zu übergeben, und behandeln Sie die Daten in diesen Methoden.

FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Regler

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    @FXML
    private Text target;

    public void setData(String data) {
        target.setText(data);
    }

}
```

Code, der zum Laden der fxml verwendet wird

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));
Parent root = loader.load();
TestController controller = loader.<TestController>getController();
controller.setData(data);
```

Daten an FXML übergeben - Angabe der Controller-Instanz

Problem: Einige Daten müssen an eine Szene übergeben werden, die aus einem Fxml geladen wurde.

Lösung

`FXMLLoader` Sie den Controller mit der `FXMLLoader` Instanz ein, die später zum Laden der fxml verwendet wird.

Stellen Sie sicher, dass der Controller die relevanten Daten enthält, bevor Sie die fxml laden.

Hinweis: In diesem Fall darf die fxml-Datei nicht das Attribut `fx:controller` enthalten.

FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Regler

```
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Code, der zum Laden der fxml verwendet wird

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

TestController controller = new TestController(data);
loader.setController(controller);

Parent root = loader.load();
```

Übergabe von Parametern an FXML - mithilfe einer ControllerFactory

Problem: Einige Daten müssen an eine Szene übergeben werden, die aus einem Fxml geladen wurde.

Lösung

Geben Sie eine Controller-Factory an, die für die Erstellung der Controller verantwortlich ist.

Übergeben Sie die Daten an die von der Fabrik erstellte Controller-Instanz.

FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

Regler

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Code, der zum Laden der fxml verwendet wird

String data = "Hallo Welt!";

```
Map<Class, Callable<?>> creators = new HashMap<>();
creators.put(TestController.class, new Callable<TestController>() {

    @Override
    public TestController call() throws Exception {
        return new TestController(data);
    }

});

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

loader.setControllerFactory(new Callback<Class<?>, Object>() {
```

```
    @Override
    public Object call(Class<?> param) {
        Callable<?> callable = creators.get(param);
        if (callable == null) {
            try {
                // default handling: use no-arg constructor
                return param.newInstance();
            } catch (InstantiationException | IllegalAccessException ex) {
                throw new IllegalStateException(ex);
            }
        } else {
            try {
                return callable.call();
            } catch (Exception ex) {
                throw new IllegalStateException(ex);
            }
        }
    }
});

Parent root = loader.load();
```

Dies mag komplex erscheinen, aber es kann nützlich sein, wenn die fxml entscheiden kann, welche Controller-Klasse sie benötigt.

Instanzerstellung in FXML

Die folgende Klasse zeigt, wie Instanzen von Klassen erstellt werden können:

JavaFX 8

Die Annotation in `Person(@NamedArg("name") String name)` muss entfernt werden, da die Annotation `@NamedArg` nicht verfügbar ist.

```
package fxml.sample;

import javafx.beans.NamedArg;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public static final Person JOHN = new Person("John");

    public Person() {
        System.out.println("Person()");
    }

    public Person(@NamedArg("name") String name) {
        System.out.println("Person(String)");
        this.name.set(name);
    }

    public Person(Person person) {
        System.out.println("Person(Person)");
        this.name.set(person.getName());
```

```
    }

    private final StringProperty name = new SimpleStringProperty();

    public final String getName() {
        System.out.println("getter");
        return this.name.get();
    }

    public final void setName(String value) {
        System.out.println("setter");
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        System.out.println("property getter");
        return this.name;
    }

    public static Person valueOf(String value) {
        System.out.println("valueOf");
        return new Person(value);
    }

    public static Person createPerson() {
        System.out.println("createPerson");
        return new Person();
    }

}
```

Angenommen, die `Person` Klasse wurde vor dem Laden der fxml bereits initialisiert.

Ein Hinweis zu den Einfuhren

Im folgenden fxml-Beispiel wird der Importabschnitt ausgelassen. Die fxml sollte jedoch mit beginnen

```
<?xml version="1.0" encoding="UTF-8"?>
```

gefolgt von einem Importabschnitt, in dem alle in der fxml-Datei verwendeten Klassen importiert werden. Diese Importe ähneln nicht-statischen Importen, werden jedoch als Verarbeitungsanweisungen hinzugefügt. **Sogar Klassen aus dem `java.lang` Paket müssen importiert werden.**

In diesem Fall sollten folgende Importe hinzugefügt werden:

```
<?import java.lang.*?>
<?import fxml.sample.Person?>
```

JavaFX 8

`@NamedArg` Konstruktor mit Anmerkungen

Wenn es einen Konstruktor gibt, bei dem jeder Parameter mit `@NamedArg` und alle Werte der `@NamedArg` Annotationen in der fxml vorhanden sind, wird der Konstruktor mit diesen Parametern verwendet.

```
<Person name="John"/>
```

```
<Person xmlns:fx="http://javafx.com/fxml">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Wenn beide geladen sind, führt dies zu folgender Konsolenausgabe:

```
Person(String)
```

Kein Argumentkonstruktor

Wenn kein geeigneter mit `@NamedArg` versehener `@NamedArg` Konstruktor verfügbar ist, wird der Konstruktor verwendet, der keine Parameter verwendet.

Entfernen Sie die Annotation `@NamedArg` aus dem Konstruktor und versuchen Sie, zu laden.

```
<Person name="John"/>
```

Dadurch wird der Konstruktor ohne Parameter verwendet.

Ausgabe:

```
Person()
setter
```

`fx:value` Wertattribut

Das Attribut `fx:value` kann verwendet werden, um seinen Wert an eine `static valueOf` Methode zu übergeben, die einen `String` Parameter verwendet und die zu verwendende Instanz `valueOf` .

Beispiel

```
<Person xmlns:fx="http://javafx.com/fxml" fx:value="John"/>
```

Ausgabe:

```
valueOf
Person(String)
```

`fx:factory`

Das Attribut `fx:factory` ermöglicht die Erstellung von Objekten mit beliebigen `static` Methoden, die keine Parameter verwenden.

Beispiel

```
<Person xmlns:fx="http://javafx.com/fxml" fx:factory="createPerson">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Ausgabe:

```
createPerson
Person()
setter
```

`<fx:copy>`

Mit `fx:copy` ein Kopierkonstruktor aufgerufen werden. Die Angabe der `fx:id` eines anderen der `source` - Attribut des Tags wird die Copykonstruktor mit diesem Objekt als Parameter aufrufen.

Beispiel:

```
<ArrayList xmlns:fx="http://javafx.com/fxml">
    <Person fx:id="p1" fx:constant="JOHN"/>
    <fx:copy source="p1"/>
</ArrayList>
```

Ausgabe

```
Person(Person)
getter
```

`fx:constant`

`fx:constant` ermöglicht das Abrufen eines Wertes aus einem `static final` Endfeld.

Beispiel

```
<Person xmlns:fx="http://javafx.com/fxml" fx:constant="JOHN"/>
```

erzeugt keine Ausgabe, da diese nur auf `JOHN` verweist, das beim Initialisieren der Klasse erstellt wurde.

Eigenschaften festlegen

Es gibt mehrere Möglichkeiten, einem Objekt in fxml Daten hinzuzufügen:

`<property>`

-Tag

Ein Tag mit dem Namen einer Eigenschaft kann als untergeordnetes Element eines Elements hinzugefügt werden, das zum Erstellen einer Instanz verwendet wird. Das untergeordnete Element dieses Tags wird der Eigenschaft mit dem Setter zugewiesen oder dem Inhalt der Eigenschaft hinzugefügt (readonly list / map properties).

Standardeigenschaft

Eine Klasse kann mit der Annotation `@DefaultProperty` kommentiert werden. In diesem Fall können Elemente direkt als untergeordnetes Element hinzugefügt werden, ohne dass ein Element mit dem Namen der Eigenschaft verwendet wird.

`property="value"` -Attribut

Eigenschaften können mit dem Eigenschaftsnamen als Attributnamen und dem Wert als Attributwert zugewiesen werden. Dies hat den gleichen Effekt wie das Hinzufügen des folgenden Elements als untergeordnetes Element des Tags:

```
<property>
    <String fx:value="value" />
</property>
```

statische Setzer

Eigenschaften können auch mit `static` Setzern eingestellt werden. Dies sind `static` Methoden mit dem Namen `setProperty` , bei denen das Element als erster Parameter und der Wert als zweiter Parameter festgelegt wird. Diese Methoden können in jeder Klasse verwendet werden und können mit `ContainingClass.property` anstelle des üblichen Eigenschaftennamens verwendet werden.

Anmerkung: Derzeit scheint es notwendig zu sein, eine entsprechende statische Getter-Methode zu haben (dh eine statische Methode mit dem Namen `getProperty` die das Element als Parameter in derselben Klasse wie der statische Setzer verwendet), damit dies funktioniert, sofern der `getProperty String` .

Typ Zwang

Der folgende Mechanismus wird verwendet, um ein Objekt der richtigen Klasse während Zuweisungen abzurufen, z. B. um den Parametertyp einer Setter-Methode anzupassen.

Wenn die Klassen zuweisbar sind, wird der Wert selbst verwendet.

Andernfalls wird der Wert wie folgt konvertiert

Zieltyp	verwendeter Wert (Quellwert `s`)
`Boolean` , `boolean`	`Boolean.valueOf(s)`
`char` , `Character`	`s.toString.charAt(0)`
anderer primitiver Typ oder Wrapper-Typ	geeignete Methode für den `valueOf(s.toString())` , falls `s` eine `Number` , `valueOf(s.toString())` für den Wrapper-Typ
`BigInteger`	`BigInteger.valueOf(s.longValue())` ist `s` ist eine `Number` , ansonsten `new BigInteger(s.toString())`
`BigDecimal`	`BigDecimal.valueOf(s.doubleValue())` ist `s` eine `Number` , ansonsten `new BigDecimal(s.toString())`
Nummer	`Double.valueOf(s.toString())` wenn `s.toString()` enthält `.` , `Long.valueOf(s.toString())` ansonsten
`Class`	`Class.forName(s.toString())` mit dem Kontext `ClassLoader` des aktuellen Threads aufgerufen, ohne die Klasse zu initialisieren
enum	Das Ergebnis der `valueOf` Methode umgewandelt zusätzlich zu einem Großbuchstaben `String` getrennt durch _ vor jeder Großbuchstabe eingefügt, wenn `s` a `String` , der mit einem Kleinbuchstaben beginnt ,
andere	Der von einer `static valueOf` Methode im targetType zurückgegebene Wert, dessen Parameter mit dem Typ `s` oder einer Superklasse dieses Typs übereinstimmt

Hinweis: Dieses Verhalten ist nicht gut dokumentiert und kann Änderungen unterliegen.

Beispiel

```
public enum Location {
    WASHINGTON_DC,
    LONDON;
}
```

```
package fxml.sample;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
import javafx.beans.DefaultProperty;

@DefaultProperty("items")
public class Sample {

    private Location loaction;

    public Location getLoaction() {
        return loaction;
    }

    public void setLoaction(Location loaction) {
        this.loaction = loaction;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    int number;

    private final List<Object> items = new ArrayList<>();

    public List<Object> getItems() {
        return items;
    }

    private final Map<String, Object> map = new HashMap<>();

    public Map<String, Object> getMap() {
        return map;
    }

    private BigInteger serialNumber;

    public BigInteger getSerialNumber() {
        return serialNumber;
    }

    public void setSerialNumber(BigInteger serialNumber) {
        this.serialNumber = serialNumber;
    }

    @Override
    public String toString() {
        return "Sample{" + "loaction=" + loaction + ", number=" + number + ", items=" + items
+ ", map=" + map + ", serialNumber=" + serialNumber + '}';
    }

}
```

```
package fxml.sample;

public class Container {

    public static int getNumber(Sample sample) {
        return sample.number;
```

```
    }

    public static void setNumber(Sample sample, int number) {
        sample.number = number;
    }

    private final String value;

    private Container(String value) {
        this.value = value;
    }

    public static Container valueOf(String s) {
        return new Container(s);
    }

    @Override
    public String toString() {
        return "42" + value;
    }

}
```

Das Ergebnis des Ladens der unten angegebenen `fxml` Datei wird `fxml`

```
Sample{loaction=WASHINGTON_DC, number=5, items=[42a, 42b, 42c, 42d, 42e, 42f], map={answer=42,
g=9.81, hello=42A, sample=Sample{loaction=null, number=33, items=[], map={},
serialNumber=null}}, serialNumber=4299}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import fxml.sample.*?>

<Sample xmlns:fx="http://javafx.com/fxml/1" Container.number="5" loaction="washingtonDc">

    <!-- set serialNumber property (type coercion) -->
    <serialNumber>
        <Container fx:value="99"/>
    </serialNumber>

    <!-- Add elements to default property-->
    <Container fx:value="a"/>
    <Container fx:value="b"/>
    <Container fx:value="c"/>
    <Container fx:value="d"/>
    <Container fx:value="e"/>
    <Container fx:value="f"/>

    <!-- fill readonly map property -->
    <map g="9.81">
        <hello>
            <Container fx:value="A"/>
        </hello>
        <answer>
            <Container fx:value=""/>
        </answer>
        <sample>
            <Sample>
```

```
                <!-- static setter-->
                <Container.number>
                    <Integer fx:value="33" />
                </Container.number>
            </Sample>
        </sample>
    </map>
</Sample>
```

FXML und Controller online lesen: https://riptutorial.com/de/javafx/topic/1580/fxml-und-controller

Kapitel 10: Internationalisierung in JavaFX

Examples

Ressourcenpaket laden

JavaFX bietet eine einfache Möglichkeit, Ihre Benutzeroberflächen zu internationalisieren. Beim Erstellen einer Ansicht aus einer FXML-Datei können Sie dem `FXMLLoader` ein Ressourcenpaket zur Verfügung stellen:

```
Locale locale = new Locale("en", "UK");
ResourceBundle bundle = ResourceBundle.getBundle("strings", locale);

Parent root = FXMLLoader.load(getClass().getClassLoader()
                                .getResource("ui/main.fxml"), bundle);
```

Dieses mitgelieferte Paket wird automatisch verwendet, um alle Texte in Ihrer FXML-Datei zu übersetzen, die mit einem `%` . Nehmen wir an, Ihre Eigenschaftsdatei `strings_en_UK.properties` enthält die folgende Zeile:

```
ui.button.text=I'm a Button
```

Wenn Sie eine Button-Definition in Ihrer FXML haben:

```
<Button text="%ui.button.text"/>
```

Es erhält automatisch die Übersetzung für den Schlüssel `ui.button.text` .

Regler

Ressourcenpakete enthalten ländereinstellungsspezifische Objekte. Sie können das Bundle während der Erstellung an den `FXMLLoader` . Der Controller muss die `Initializable` Schnittstelle implementieren und die `initialize(URL location, ResourceBundle resources)` überschreiben. Der zweite Parameter dieser Methode ist `ResourceBundle` , das vom FXMLLoader an den Controller übergeben wird und von dem Controller verwendet werden kann, um Texte weiter zu übersetzen oder andere vom Gebietsschema abhängige Informationen zu ändern.

```
public class MyController implements Initializable {

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        label.setText(resources.getString("country"));
    }
}
```

Dynamisches Wechseln der Sprache, wenn die Anwendung ausgeführt wird

Dieses Beispiel zeigt, wie eine JavaFX-Anwendung erstellt wird, bei der die Sprache dynamisch gewechselt werden kann, während die Anwendung ausgeführt wird.

Dies sind die Nachrichtenpaketdateien, die im Beispiel verwendet werden:

messages_de.properties :

```
window.title=Dynamic language change
button.english=English
button.german=German
label.numSwitches=Number of language switches: {0}
```

messages_de.properties :

```
window.title=Dynamischer Sprachwechsel
button.english=Englisch
button.german=Deutsch
label.numSwitches=Anzahl Sprachwechsel: {0}
```

Die Grundidee ist die Verwendung einer Dienstklasse I18N (alternativ kann dies als Singleton implementiert werden).

```
import javafx.beans.binding.Bindings;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.concurrent.Callable;

/**
 * I18N utility class..
 */
public final class I18N {

    /** the current selected Locale. */
    private static final ObjectProperty<Locale> locale;

    static {
        locale = new SimpleObjectProperty<>(getDefaultLocale());
        locale.addListener((observable, oldValue, newValue) -> Locale.setDefault(newValue));
    }

    /**
     * get the supported Locales.
     *
     * @return List of Locale objects.
     */
    public static List<Locale> getSupportedLocales() {
        return new ArrayList<>(Arrays.asList(Locale.ENGLISH, Locale.GERMAN));
```

```
    }

    /**
     * get the default locale. This is the systems default if contained in the supported
locales, english otherwise.
     *
     * @return
     */
    public static Locale getDefaultLocale() {
        Locale sysDefault = Locale.getDefault();
        return getSupportedLocales().contains(sysDefault) ? sysDefault : Locale.ENGLISH;
    }

    public static Locale getLocale() {
        return locale.get();
    }

    public static void setLocale(Locale locale) {
        localeProperty().set(locale);
        Locale.setDefault(locale);
    }

    public static ObjectProperty<Locale> localeProperty() {
        return locale;
    }

    /**
     * gets the string with the given key from the resource bundle for the current locale and
uses it as first argument
     * to MessageFormat.format, passing in the optional args and returning the result.
     *
     * @param key
     *         message key
     * @param args
     *         optional arguments for the message
     * @return localized formatted string
     */
    public static String get(final String key, final Object... args) {
        ResourceBundle bundle = ResourceBundle.getBundle("messages", getLocale());
        return MessageFormat.format(bundle.getString(key), args);
    }

    /**
     * creates a String binding to a localized String for the given message bundle key
     *
     * @param key
     *         key
     * @return String binding
     */
    public static StringBinding createStringBinding(final String key, Object... args) {
        return Bindings.createStringBinding(() -> get(key, args), locale);
    }

    /**
     * creates a String Binding to a localized String that is computed by calling the given
func
     *
     * @param func
     *         function called on every change
     * @return StringBinding
     */
```

```
    public static StringBinding createStringBinding(Callable<String> func) {
        return Bindings.createStringBinding(func, locale);
    }

    /**
     * creates a bound Label whose value is computed on language change.
     *
     * @param func
     *         the function to compute the value
     * @return Label
     */
    public static Label labelForValue(Callable<String> func) {
        Label label = new Label();
        label.textProperty().bind(createStringBinding(func));
        return label;
    }

    /**
     * creates a bound Button for the given resourcebundle key
     *
     * @param key
     *         ResourceBundle key
     * @param args
     *         optional arguments for the message
     * @return Button
     */
    public static Button buttonForKey(final String key, final Object... args) {
        Button button = new Button();
        button.textProperty().bind(createStringBinding(key, args));
        return button;
    }
}
```

Diese Klasse hat ein statisches Feld `locale` , das ein Java ist `Locale` Objekt in einer JavaFX gewickelte `ObjectProperty` , so dass Bindungen können für dieses Objekt erstellt werden. Die ersten Methoden sind die Standardmethoden zum Abrufen und Festlegen einer JavaFX-Eigenschaft.

Das `get(final String key, final Object... args)` ist die Kernmethode, die für die eigentliche Extraktion einer Nachricht aus einem `ResourceBundle` .

Die beiden Methoden `createStringBinding` erstellen eine `StringBinding` , die an das `locale` gebunden ist. `StringBinding` ändern sich die Bindungen, wenn sich die Eigenschaft " `locale` ändert. Der erste verwendet seine Argumente, um eine Nachricht mithilfe der oben genannten `get` Methode abzurufen und zu formatieren. Der zweite wird in einem `Callable` , das den neuen String-Wert erzeugen muss.

Die letzten beiden Methoden sind Methoden zum Erstellen von JavaFX-Komponenten. Die erste Methode wird zum Erstellen eines `Label` und verwendet `Callable` für die interne Zeichenfolgenbindung. Die zweite erstellt einen `Button` und verwendet einen Schlüsselwert zum Abrufen der String-Bindung.

Natürlich könnten noch viele andere Objekte wie `MenuItem` oder `ToolTip` aber diese beiden sollten für ein Beispiel ausreichen.

Dieser Code zeigt, wie diese Klasse in der Anwendung verwendet wird:

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

import java.util.Locale;

/**
 * Sample application showing dynamic language switching,
 */
public class I18nApplication extends Application {

    /** number of language switches. */
    private Integer numSwitches = 0;

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

        primaryStage.titleProperty().bind(I18N.createStringBinding("window.title"));

        // create content
        BorderPane content = new BorderPane();

        // at the top two buttons
        HBox hbox = new HBox();
        hbox.setPadding(new Insets(5, 5, 5, 5));
        hbox.setSpacing(5);

        Button buttonEnglish = I18N.buttonForKey("button.english");
        buttonEnglish.setOnAction((evt) -> switchLanguage(Locale.ENGLISH));
        hbox.getChildren().add(buttonEnglish);

        Button buttonGerman = I18N.buttonForKey("button.german");
        buttonGerman.setOnAction((evt) -> switchLanguage(Locale.GERMAN));
        hbox.getChildren().add(buttonGerman);

        content.setTop(hbox);

        // a label to display the number of changes, recalculating the text on every change
        final Label label = I18N.labelForValue(() -> I18N.get("label.numSwitches",
numSwitches));
        content.setBottom(label);

        primaryStage.setScene(new Scene(content, 400, 200));
        primaryStage.show();
    }

    /**
     * sets the given Locale in the I18N class and keeps count of the number of switches.
     *
     * @param locale
     *         the new local to set
     */
    private void switchLanguage(Locale locale) {
        numSwitches++;
```

```
            I18N.setLocale(locale);
        }
    }
```

Die Anwendung zeigt drei verschiedene Arten der Verwendung der von der `I18N` Klasse erstellten `StringBinding` :

1. Der Fenstertitel wird direkt mit einer `StringBinding` gebunden.
2. Die Schaltflächen verwenden die Hilfsmethode mit den Nachrichtentasten
3. Das Label verwendet die `Callable` mit einem `Callable` . Dieses `Callable` verwendet die `I18N.get()` Methode, um eine formatierte übersetzte Zeichenfolge mit der tatsächlichen Anzahl der Schalter `I18N.get()` .

Wenn Sie auf eine Schaltfläche klicken, wird der Zähler erhöht und die locale-Eigenschaft von `I18N` wird festgelegt. `I18N` wird die Änderung der Zeichenfolgenbindungen ausgelöst und die Zeichenfolge der Benutzeroberfläche auf neue Werte gesetzt.

Internationalisierung in JavaFX online lesen:
https://riptutorial.com/de/javafx/topic/5434/internationalisierung-in-javafx

Kapitel 11: JavaFX-Bindungen

Examples

Einfache Eigenschaftsbindung

JavaFX verfügt über eine Bindungs-API, über die eine Eigenschaft an die andere gebunden werden kann. Das bedeutet, dass bei jeder Änderung eines Eigenschaftswerts der Wert der gebundenen Eigenschaft automatisch aktualisiert wird. Ein Beispiel für die einfache Bindung:

```
SimpleIntegerProperty first =new SimpleIntegerProperty(5); //create a property with value=5
SimpleIntegerProperty second=new SimpleIntegerProperty();

public void test()
{
    System.out.println(second.get()); // '0'
    second.bind(first);               //bind second property to first
    System.out.println(second.get()); // '5'
    first.set(16);                    //set first property's value
    System.out.println(second.get()); // '16' - the value was automatically updated
}
```

Sie können eine primitive Eigenschaft auch binden, indem Sie eine Addition, eine Subtraktion, eine Division usw. anwenden:

```
public void test2()
{
        second.bind(first.add(100));
        System.out.println(second.get()); //'105'
        second.bind(first.subtract(50));
        System.out.println(second.get()); //'-45'
}
```

Jedes Objekt kann in SimpleObjectProperty abgelegt werden:

```
SimpleObjectProperty<Color> color=new SimpleObjectProperty<>(Color.web("45f3d1"));
```

Es ist möglich, bidirektionale Bindungen zu erstellen. In diesem Fall hängen die Eigenschaften voneinander ab.

```
public void test3()
{
        second.bindBidirectional(first);
        System.out.println(second.get()+" "+first.get());
        second.set(1000);
        System.out.println(second.get()+" "+first.get()); //both are '1000'
}
```

JavaFX-Bindungen online lesen: https://riptutorial.com/de/javafx/topic/7014/javafx-bindungen

Kapitel 12: Layouts

Examples

StackPane

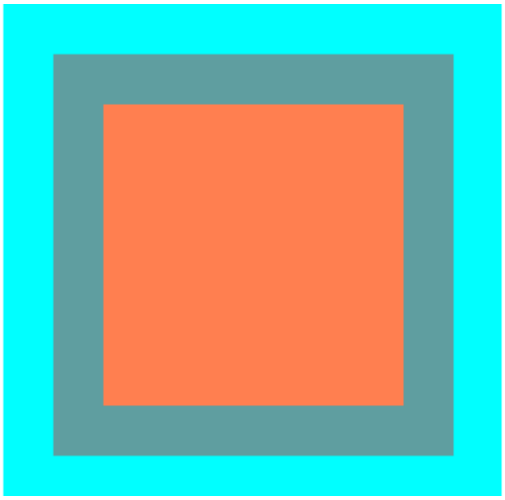
`StackPane` legt seine `StackPane` in einem Stapel von vorne nach hinten ab.

Die z-Reihenfolge der `getChildren` wird durch die Reihenfolge der `getChildren` Liste definiert (`getChildren` durch Aufruf von `getChildren`): Das `getChildren` ist das unterste und letzte `getChildren` des Stapels.

Das Stack-Fenster versucht, die Größe jedes untergeordneten Objekts so zu ändern, dass es seinen eigenen Inhaltsbereich ausfüllt. Wenn die Größe eines Kindes nicht `StackPane` kann, um den Bereich der `StackPane` zu füllen (entweder weil die Größe nicht geändert wurde oder weil die maximale Größe nicht möglich war), wird es innerhalb des Bereichs mit der `alignmentProperty` Eigenschaft der Stackpane ausgerichtet `Pos.CENTER`.

Beispiel

```
// Create a StackPane
StackPane pane = new StackPane();

// Create three squares
Rectangle rectBottom = new Rectangle(250, 250);
rectBottom.setFill(Color.AQUA);
Rectangle rectMiddle = new Rectangle(200, 200);
rectMiddle.setFill(Color.CADETBLUE);
Rectangle rectUpper = new Rectangle(150, 150);
rectUpper.setFill(Color.CORAL);

// Place them on top of each other
pane.getChildren().addAll(rectBottom, rectMiddle, rectUpper);
```



HBox und VBox

Die Layouts von `HBox` und `VBox` sind sehr ähnlich, beide legen ihre Kinder in einer einzigen Zeile an.

Gemeinsamkeiten

Wenn für eine `HBox` oder eine `VBox` ein Rand und / oder ein `VBox` festgelegt ist, werden die Inhalte innerhalb dieser Insets angeordnet.

Sie legen jedes verwaltete Kind unabhängig vom sichtbaren Eigenschaftswert des Kindes fest. Nicht verwaltete Kinder werden ignoriert.

Die Ausrichtung des Inhalts wird durch die Ausrichtungseigenschaft gesteuert, die standardmäßig auf **`Pos.TOP_LEFT`** .

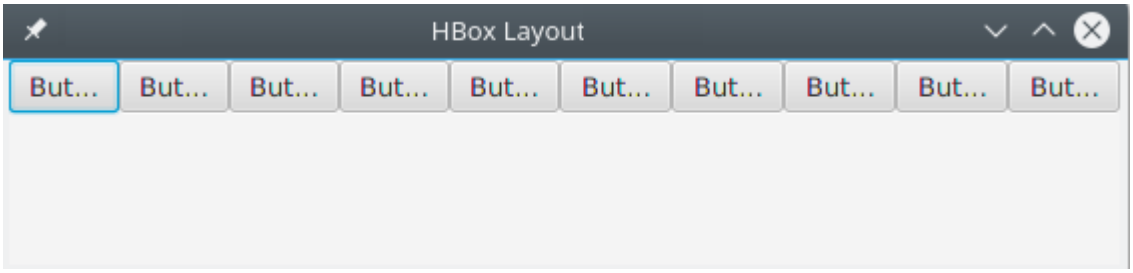
HBox

`HBox` legt ihre Kinder in einer einzigen horizontalen Reihe von links nach rechts an.

`HBox` die Größe der untergeordneten `HBox` (wenn `HBox` in der Größe verändert wird) **auf die bevorzugte Breite** s und verwendet die Eigenschaft fillHeight, um zu bestimmen, ob die Größe der Höhe geändert werden soll, um ihre eigene Höhe zu füllen oder ihre Höhe auf die bevorzugte Höhe zu setzen (Standardeinstellung für fillHeight auf true).

Erstellen einer HBox

```
// HBox example
HBox row = new HBox();
Label first = new Label("First");
Label second = new Label("Second");
row.getChildren().addAll(first, second);
```



VBox

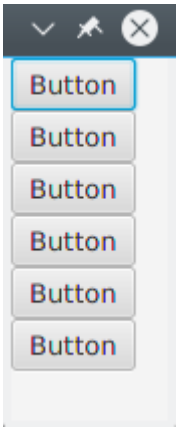
`VBox` die `VBox` in einer einzigen vertikalen Spalte von oben nach unten an.

`VBox` die Größe von untergeordneten `VBox` (falls `VBox` in der Größe geändert werden kann) **auf ihre bevorzugten Höhen** und verwendet die Eigenschaft fillWidth, um zu bestimmen, ob die Breite geändert werden soll, um ihre eigene Breite auszufüllen, oder ob sie ihre bevorzugte Breite hat (fillWidth ist standardmäßig auf true gesetzt).

Erstellen einer VBox

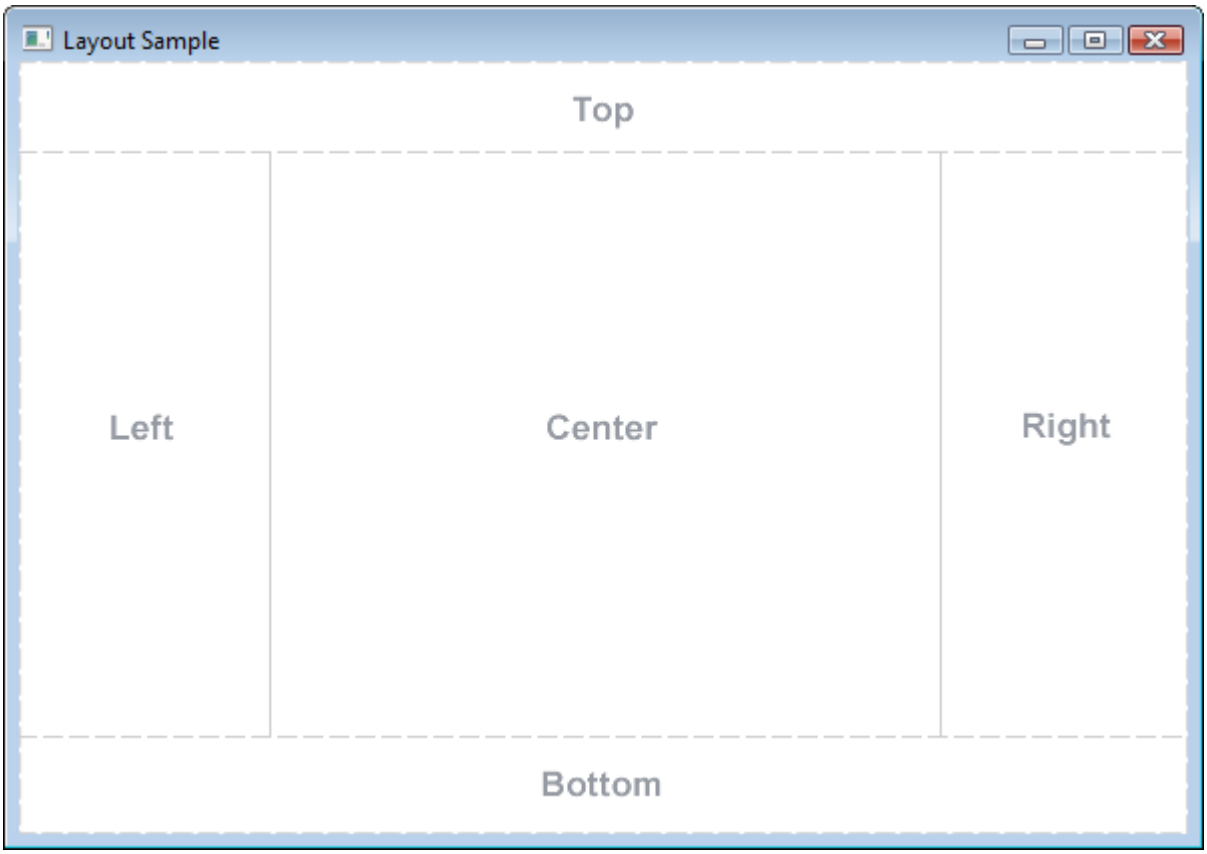
```
// VBox example
VBox column = new VBox();
```

```
Label upper = new Label("Upper");
Label lower = new Label("Lower");
column.getChildren().addAll(upper, lower);
```



BorderPane

Das `BorderPane` ist in fünf verschiedene Bereiche unterteilt.



Die Randbereiche (`Top` , `Right` , `Bottom` , `Left`) haben eine bevorzugte Größe basierend auf ihrem Inhalt. Standardmäßig nehmen sie nur das, was sie benötigen, während der `Center` Bereich den verbleibenden Speicherplatz beansprucht. Wenn die Randbereiche leer sind, nehmen sie keinen Platz ein.

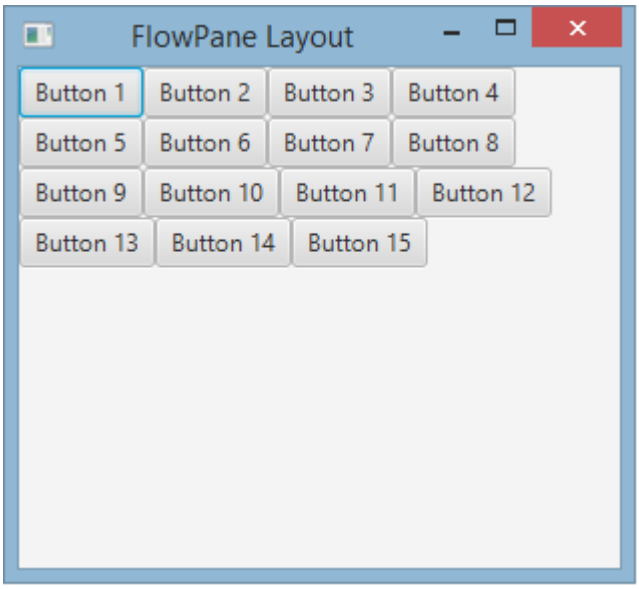
Jeder Bereich kann nur ein Element enthalten. Es kann mit den Methoden `setTop(Node)` , `setRight(Node)` , `setBottom(Node)` , `setLeft(Node)` , `setCenter(Node)` . Sie können andere Layouts verwenden, um mehrere Elemente in einem einzigen Bereich zu platzieren.

```
//BorderPane example
BorderPane pane = new BorderPane();

Label top = new Label("Top");

Label right = new Label("Right");

HBox bottom = new HBox();
bottom.getChildren().addAll(new Label("First"), new Label("Second"));

VBox left = new VBox();
left.getChildren().addAll(new Label("Upper"), new Label("Lower"));

StackPane center = new StackPane();
center.getChildren().addAll(new Label("Lorem"), new Label("ipsum"));

pane.setTop(top);        //The text "Top"
pane.setRight(right);    //The text "Right"
pane.setBottom(bottom);  //Row of two texts
pane.setLeft(left);      //Column of two texts
pane.setCenter(center);  //Two texts on each other
```

FlowPane

`FlowPane` Knoten in Zeilen oder Spalten basierend auf dem verfügbaren horizontalen oder vertikalen Platz ein. Knoten werden in die nächste Zeile umgebrochen, wenn der horizontale Abstand geringer als die Gesamtbreite aller Knoten ist. Knoten werden in die nächste Spalte eingebettet, wenn der vertikale Abstand geringer als die Summe der Knoten aller Knoten ist. Dieses Beispiel veranschaulicht das horizontale Standardlayout:

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
        FlowPane root = new FlowPane();
        for (int i=1; i<=15; i++) {
            Button b1=new Button("Button "+String.valueOf(i));
            root.getChildren().add(b1); //for adding button to root
        }
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("FlowPane Layout");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Standard- `FlowPane` Konstruktor:

```
FlowPane root = new FlowPane();
```

Zusätzliche `FlowPane` Konstruktoren:

```
FlowPane() //Creates a horizontal FlowPane layout with hgap/vgap = 0 by default.
FlowPane(double hgap, double gap) //Creates a horizontal FlowPane layout with the specified
hgap/vgap.
FlowPane(double hgap, double vgap, Node... children) //Creates a horizontal FlowPane layout
with the specified hgap/vgap.
FlowPane(Node... children) //Creates a horizontal FlowPane layout with hgap/vgap = 0.
FlowPane(Orientation orientation) //Creates a FlowPane layout with the specified orientation
and hgap/vgap = 0.
FlowPane(Orientation orientation, double hgap, double gap) //Creates a FlowPane layout with
the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, double hgap, double vgap, Node... children) //Creates a
FlowPane layout with the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, Node... children) //Creates a FlowPane layout with the
specified orientation and hgap/vgap = 0.
```

Beim Hinzufügen von Knoten zum Layout werden die Methoden `add()` oder `addAll()` des übergeordneten `Pane` :

```
Button btn = new Button("Demo Button");
root.getChildren().add(btn);
root.getChildren().addAll(…);
```

Ein `FlowPane` Knoten standardmäßig von links nach rechts an. Um die `setAlignment()` zu ändern, rufen Sie die `setAlignment()` Methode auf, indem Sie einen Aufzählungswert vom Typ `Pos` .

Einige häufig verwendete Flussanpassungen:

```
root.setAlignment(Pos.TOP_RIGHT);     //for top right
root.setAlignment(Pos.TOP_CENTER);    //for top Center
root.setAlignment(Pos.CENTER);        //for Center
root.setAlignment(Pos.BOTTOM_RIGHT);  //for bottom right
```

`GridPane` die `GridPane` in einem flexiblen Raster aus Zeilen und Spalten an.

Kinder der GridPane

Ein `GridPane` kann an einer beliebigen Stelle innerhalb der `GridPane` platziert werden und kann sich über mehrere Zeilen / Spalten erstrecken (die Standardspanne ist 1). Die Platzierung innerhalb des Gitters wird durch die Layouteinschränkungen definiert:

Zwang	Beschreibung
columnIndex	Spalte, in der der Layoutbereich des Kindes beginnt.
rowIndex	Zeile, in der der Layoutbereich des Kindes beginnt.
columnSpan	Die Anzahl der Spalten, die sich der Layoutbereich des Kindes erstreckt, erstreckt sich horizontal.
rowSpan	Die Anzahl der Zeilen, die der Layoutbereich des Kindes vertikal umfasst.

Die Gesamtzahl der Zeilen / Spalten muss nicht vorab angegeben werden, da der Raster automatisch das Raster erweitert / verkleinert, um den Inhalt aufzunehmen.

Hinzufügen von Kindern zum GridPane

Um neue `Node` s zu einem `GridPane` hinzuzufügen, `GridPane` die **Layouteinschränkungen** für die `GridPane` mithilfe der statischen Methode der `GridPane` Klasse `GridPane` werden. `GridPane` können dann einer `GridPane` Instanz `GridPane` werden.

```
GridPane gridPane = new GridPane();

// Set the constraints: first row and first column
Label label = new Label("Example");
GridPane.setRowIndex(label, 0);
GridPane.setColumnIndex(label, 0);
// Add the child to the grid
gridpane.getChildren().add(label);
```

`GridPane` bietet praktische Methoden zum Kombinieren dieser Schritte:

```
gridPane.add(new Button("Press me!"), 1, 0); // column=1 row=0
```

Die `GridPane` Klasse bietet auch statische Setter-Methoden zum **Festlegen** des **Zeilen- und Spaltenbereichs** von **untergeordneten** Elementen:

```
Label labelLong = new Label("Its a long text that should span several rows");
```

```
GridPane.setColumnSpan(labelLong, 2);
gridPane.add(labelLong, 0, 1);  // column=0 row=1
```

Größe der Spalten und Zeilen

Standardmäßig werden Zeilen und Spalten entsprechend ihrem Inhalt angepasst. Im Falle der Notwendigkeit der **expliziten Kontrolle der Zeilen- und Spaltengrößen,** `RowConstraints` und `ColumnConstraints` können Instanzen auf die hinzugefügt werden `GridPane` . Durch das Hinzufügen dieser beiden Bedingungen wird die Größe des obigen Beispiels so geändert, dass die erste Spalte 100 Pixel und die zweite Spalte 200 Pixel lang ist.

```
gridPane.getColumnConstraints().add(new ColumnConstraints(100));
gridPane.getColumnConstraints().add(new ColumnConstraints(200));
```

Standardmäßig `GridPane` das `GridPane` die Zeilen / Spalten auf ihre bevorzugten Größen, auch wenn die Größe des Gridpane größer ist als seine bevorzugte. Um **dynamische Spalten- /** Zeilengrößen zu unterstützen, bietet die Klasse contstaints drei Eigenschaften: Mindestgröße, Höchstgröße und bevorzugte Größe.

Zusätzlich `ColumnConstraints` bietet `setHGrow` und `RowConstraints` bietet `setVGrow` Methoden , um **die Priorität der wachsenden und schrumpf** zu **beeinflussen.** Die drei vordefinierten Prioritäten sind:

- **Priority.ALWAYS** : Versuchen Sie immer, zu wachsen (oder zu schrumpfen), und teilen Sie die Vergrößerung (oder Abnahme) des Speicherbereichs mit anderen Layoutbereichen, die einen Anstieg (oder ein Verkleinern) von IMMER aufweisen
- **Priority.SOMETIMES** : Wenn es keine anderen Layoutbereiche gibt, in denen der Wert für Wachsen (oder Verkleinern) auf ALWAYS gesetzt ist oder diese Layoutbereiche nicht den gesamten vergrößerten (oder verringerten) Speicherplatz absorbiert haben, teilen sich die Speicherplatzerhöhungen (oder -abnahmen) mit andere Layoutbereiche von SOMETIMES.
- **Priority.NEVER** : Der Layoutbereich wird niemals vergrößert (oder verkleinert), wenn der verfügbare Platz in der Region zunimmt (oder abnimmt).

```
ColumnConstraints column1 = new ColumnConstraints(100, 100, 300);
column1.setHgrow(Priority.ALWAYS);
```

Die oben definierte Spalte hat eine minimale Größe von 100 Pixeln und versucht immer zu wachsen, bis sie die maximale Breite von 300 Pixeln erreicht.

Es ist auch möglich, die **prozentuale Größenänderung** für Zeilen und Spalten festzulegen. Im folgenden Beispiel wird ein `GridPane` definiert, `GridPane` dem die erste Spalte 40% der Breite des Rasters ausfüllt, die zweite Spalte die 60%.

```
GridPane gridpane = new GridPane();
ColumnConstraints column1 = new ColumnConstraints();
column1.setPercentWidth(40);
ColumnConstraints column2 = new ColumnConstraints();
```

```
column2.setPercentWidth(60);
gridpane.getColumnConstraints().addAll(column1, column2);
```

Ausrichtung von Elementen innerhalb der Gitterzellen

Die Ausrichtung von `Node` s kann mithilfe der `setHalignment` (horizontal) -Methode der `ColumnConstraints` Klasse und der `setValignment` (Vertical) -Methode der `RowConstraints` Klasse definiert werden.

```
ColumnConstraints column1 = new ColumnConstraints();
column1.setHalignment(HPos.RIGHT);

RowConstraints row1 = new RowConstraints();
row1.setValignment(VPos.CENTER);
```

TilePane

Das Layout der Kachelfläche ist dem FlowPane-Layout ähnlich. TilePane platziert alle Knoten in einem Raster, in dem jede Zelle oder Kachel dieselbe Größe hat. Es ordnet Knoten in sauberen Zeilen und Spalten an, entweder horizontal oder vertikal.

```
    import javafx.application.Application;
    import javafx.scene.Scene;
    import javafx.scene.control.Button;
    import javafx.scene.layout.TilePane;
    import javafx.stage.Stage;

    public class Main extends Application {

        @Override
        public void start(Stage primaryStage) {
            primaryStage.setTitle("TilePane Demo");
            double width = 400;
            double height = 300;
            TilePane root = new TilePane();
            root.setStyle("-fx-background-color:blue");
            // to set horizontal and vertical gap
            root.setHgap(20);
            root.setVgap(50);
            Button bl = new Button("Buttons");
            root.getChildren().add(bl);
            Button btn = new Button("Button");
            root.getChildren().add(btn);
            Button btn1 = new Button("Button 1");
            root.getChildren().add(btn1);
            Button btn2 = new Button("Button 2");
            root.getChildren().add(btn2);
            Button btn3 = new Button("Button 3");
            root.getChildren().add(btn3);
            Button btn4 = new Button("Button 4");
            root.getChildren().add(btn4);

            Scene scene = new Scene(root, width, height);
            primaryStage.setScene(scene);
            primaryStage.show();
```

```
        }


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

Ausgabe

Tilepane erstellen

```
TilePane root = new TilePane();
```

setHgap () Die Methode setVgap () wird verwendet, um eine Lücke zwischen Spalte und Spalte herzustellen. Wir können auch die Spalten für das Layout festlegen, indem Sie verwenden

```
int columnCount = 2;
root.setPrefColumns(columnCount);
```

AnchorPane

`AnchorPane` a ist ein Layout, mit dem der Inhalt in einem bestimmten Abstand von den Seiten platziert werden kann.

Es gibt 4 Methoden zum Einstellen und 4 Methoden zum `AnchorPane` der Abstände in `AnchorPane` . Der erste Parameter dieser Methoden ist der untergeordnete `Node` . Der zweite Parameter der Setter ist der zu verwendende `Double` Wert. Dieser Wert kann `null` , um keine Einschränkung für die angegebene Seite anzuzeigen.

Setter-Methode	Getter-Methode
setBottomAnchor	getBottomAnchor
setLeftAnchor	getLeftAnchor
setRightAnchor	getRightAnchor
setTopAnchor	getTopAnchor

Im folgenden Beispiel werden Knoten in bestimmten Abständen von den Seiten platziert.

Der `center` Bereich wird ebenfalls angepasst, um die angegebenen Abstände von den Seiten einzuhalten. Beobachten Sie das Verhalten, wenn die Größe des Fensters geändert wird.

```
public static void setBackgroundColor(Region region, Color color) {
    // change to 50% opacity
    color = color.deriveColor(0, 1, 1, 0.5);
    region.setBackground(new Background(new BackgroundFill(color, CornerRadii.EMPTY,
Insets.EMPTY)));
}

@Override
public void start(Stage primaryStage) {
    Region right = new Region();
    Region top = new Region();
    Region left = new Region();
    Region bottom = new Region();
    Region center = new Region();

    right.setPrefSize(50, 150);
    top.setPrefSize(150, 50);
    left.setPrefSize(50, 150);
    bottom.setPrefSize(150, 50);

    // fill with different half-transparent colors
    setBackgroundColor(right, Color.RED);
    setBackgroundColor(left, Color.LIME);
    setBackgroundColor(top, Color.BLUE);
    setBackgroundColor(bottom, Color.YELLOW);
    setBackgroundColor(center, Color.BLACK);

    // set distances to sides
    AnchorPane.setBottomAnchor(bottom, 50d);
    AnchorPane.setTopAnchor(top, 50d);
    AnchorPane.setLeftAnchor(left, 50d);
    AnchorPane.setRightAnchor(right, 50d);

    AnchorPane.setBottomAnchor(center, 50d);
    AnchorPane.setTopAnchor(center, 50d);
    AnchorPane.setLeftAnchor(center, 50d);
    AnchorPane.setRightAnchor(center, 50d);

    // create AnchorPane with specified children
    AnchorPane anchorPane = new AnchorPane(left, top, right, bottom, center);

    Scene scene = new Scene(anchorPane, 200, 200);
```

```
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Layouts online lesen: https://riptutorial.com/de/javafx/topic/2121/layouts

Kapitel 13: Radio knopf

Examples

Optionsfelder erstellen

Optionsfelder ermöglichen es dem Benutzer, ein Element aus den angegebenen auszuwählen. Es gibt zwei Möglichkeiten, einen `RadioButton` mit einem Text daneben zu deklarieren. Entweder indem Sie den Standardkonstruktor `RadioButton()` und den Text mit der Methode `setText(String)` oder den anderen Konstruktor `RadioButton(String)` .

```
RadioButton radioButton1 = new RadioButton();
radioButton1.setText("Select me!");
RadioButton radioButton2= new RadioButton("Or me!");
```

Als `RadioButton` eine Erweiterung ist `Labeled` kann es auch ein sein `Image` mit dem angegebenen `RadioButton` . Nachdem Sie den `RadioButton` mit einem der Konstruktoren erstellt haben, fügen Sie das `Image` einfach mit der `setGraphic(ImageView)` -Methode wie hier hinzu:

```
Image image = new Image("ok.jpg");
RadioButton radioButton = new RadioButton("Agree");
radioButton.setGraphic(new ImageView(image));
```

Verwenden Sie Gruppen für Optionsfelder

Eine `ToggleGroup` wird zum Verwalten der `RadioButton` s verwendet, sodass jeweils nur einer in jeder Gruppe ausgewählt werden kann.

Erstellen Sie eine einfache `ToggleGroup` wie folgt:

```
ToggleGroup group = new ToggleGroup();
```

Nachdem Sie eine `Togglegroup` haben, kann sie mit `setToggleGroup(ToggleGroup)` den `RadioButton` `setToggleGroup(ToggleGroup)` . Verwenden Sie `setSelected(Boolean)` , um einen der `RadioButton` s `RadioButton` .

```
RadioButton radioButton1 = new RadioButton("stackoverlow is awesome! :)");
radioButton1.setToggleGroup(group);
radioButton1.setSelected(true);

RadioButton radioButton2 = new RadioButton("stackoverflow is ok :|");
radioButton2.setToggleGroup(group);

RadioButton radioButton3 = new RadioButton("stackoverflow is useless :(");
radioButton3.setToggleGroup(group);
```

Ereignisse für Radioknöpfe

Normalerweise führt die Anwendung eine Aktion aus, wenn einer der `RadioButton` s in einer `ToggleGroup` ausgewählt wird. Nachfolgend ein Beispiel, in dem die Benutzerdaten des ausgewählten `RadioButton` die mit `setUserData(Object)` .

```
radioButton1.setUserData("awesome")
radioButton2.setUserData("ok");
radioButton3.setUserData("useless");

ToggleGroup group = new ToggleGroup();
group.selectedToggleProperty().addListener((obserableValue, old_toggle, new_toggle) -> {
    if (group.getSelectedToggle() != null) {
        System.out.println("You think that stackoverflow is " +
group.getSelectedToggle().getUserData().toString());
    }
});
```

Anfordern des Fokus für Optionsfelder

Nehmen wir an, der zweite von `RadioButton` ist mit `setSelected(Boolean)` . Der Fokus befindet sich standardmäßig noch auf dem ersten `RadioButton` . Um dies zu ändern, verwenden Sie die `requestFocus()` -Methode.

```
radioButton2.setSelected(true);
radioButton2.requestFocus();
```

Radio knopf online lesen: https://riptutorial.com/de/javafx/topic/5906/radio-knopf

Kapitel 14: ScrollPane

Einführung

Das ScrollPane ist ein Steuerelement, das eine dynamische Ansicht des Inhalts bietet. Diese Ansicht wird auf verschiedene Arten gesteuert. (Inkrement-Dekrement-Taste / Mausrad), um eine integrierte Ansicht des Inhalts zu erhalten.

Examples

A) Feste Inhaltsgröße:

Die Größe des Inhalts entspricht der des ScrollPane-Containers.

```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane(content);  //Initialize and add content as a parameter
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane

scrollpane.setFitToWidth(true);  //Adapt the content to the width of ScrollPane
scrollpane.setFitToHeight(true); //Adapt the content to the height of ScrollPane


scrollpane.setHbarPolicy(ScrollBarPolicy.ALWAYS);  //Control the visibility of the Horizontal
ScrollBar
scrollpane.setVbarPolicy(ScrollBarPolicy.NEVER);  //Control the visibility of the Vertical
ScrollBar
//There are three types of visibility (ALWAYS/AS_NEEDED/NEVER)
```

B) Größe des dynamischen Inhalts:

Die Größe des Inhalts ändert sich in Abhängigkeit von den hinzugefügten Elementen, die die Inhaltsgrenzen in beiden Achsen (horizontal und vertikal) überschreiten, die durch Bewegung durch die Ansicht sichtbar werden.

```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane();
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane
content.setMinSize(300,300); //Here a minimum size is set so that the container can be
extended.
```

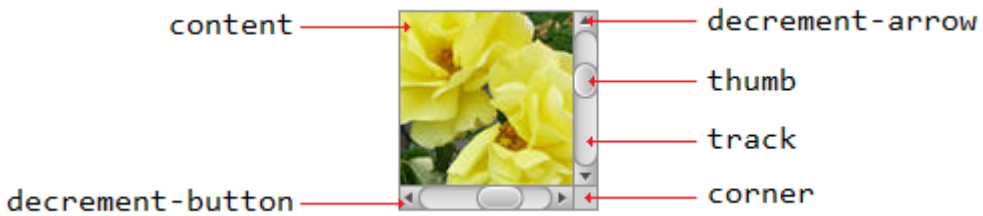
```
scrollpane.setContent(content); // we add the content to the ScrollPane
```

Hinweis: Hier brauchen wir nicht beide Methoden (setFitToWidth / setFitToHeight).

Das ScrollPane gestalten:

Das Erscheinungsbild des ScrollPane kann leicht geändert werden, indem einige Begriffe von " *CSS* " verwendet werden und einige " *Eigenschaften* " der Kontrolle beachtet werden und natürlich etwas " *Phantasie* ".

A) Die Elemente, aus denen ScrollPane besteht:



B) CSS-Eigenschaften:

```
.scroll-bar:vertical .track{}

.scroll-bar:horizontal .track{}

.scroll-bar:horizontal .thumb{}

.scroll-bar:vertical .thumb{}


.scroll-bar:vertical *.increment-button,
.scroll-bar:vertical *.decrement-button{}

.scroll-bar:vertical *.increment-arrow .content,
.scroll-bar:vertical *.decrement-arrow .content{}

.scroll-bar:vertical *.increment-arrow,
.scroll-bar:vertical *.decrement-arrow{}


.scroll-bar:horizontal *.increment-button,
.scroll-bar:horizontal *.decrement-button{}

.scroll-bar:horizontal *.increment-arrow .content,
.scroll-bar:horizontal *.decrement-arrow .content{}

.scroll-bar:horizontal *.increment-arrow,
.scroll-bar:horizontal *.decrement-arrow{}

.scroll-pane .corner{}

.scroll-pane{}
```

ScrollPane online lesen: https://riptutorial.com/de/javafx/topic/8259/scrollpane

Kapitel 15: Segeltuch

Einführung

Ein `Canvas` ist ein JavaFX- `Node` , der als leerer rechteckiger Bereich dargestellt wird und Bilder, Formen und Text anzeigen kann. Jedes `Canvas` enthält genau ein `GraphicsContext` Objekt, das für den Empfang und die Pufferung der Zeichnungsaufrufe verantwortlich ist, die am Ende von `Canvas` auf dem Bildschirm dargestellt werden.

Examples

Grundformen

`GraphicsContext` bietet eine Reihe von Methoden zum Zeichnen und Füllen von geometrischen Formen. Normalerweise müssen für diese Methoden Koordinaten als Parameter übergeben werden, entweder direkt oder in Form eines Arrays von `double` . Die Koordinaten sind immer relativ zum `Canvas` , dessen Ursprung sich in der oberen linken Ecke befindet.

Hinweis: `GraphicsContext` zeichnet nicht außerhalb der `Canvas` Grenzen, dh wenn Sie versuchen, außerhalb des durch seine Größe definierten `Canvas` Bereichs zu zeichnen und dessen Größe anschließend zu ändern, führt dies zu keinem Ergebnis.

Das folgende Beispiel zeigt, wie drei halbtransparente gefüllte geometrische Formen gezeichnet werden, die mit einem schwarzen Strich umrandet sind.

```
Canvas canvas = new Canvas(185, 70);
GraphicsContext gc = canvas.getGraphicsContext2D();

// Set stroke color, width, and global transparency
gc.setStroke(Color.BLACK);
gc.setLineWidth(2d);
gc.setGlobalAlpha(0.5d);

// Draw a square
gc.setFill(Color.RED);
gc.fillRect(10, 10, 50, 50);
gc.strokeRect(10, 10, 50, 50);

// Draw a triangle
gc.setFill(Color.GREEN);
gc.fillPolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);
gc.strokePolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);

// Draw a circle
gc.setFill(Color.BLUE);
gc.fillOval(130, 10, 50, 50);
gc.strokeOval(130, 10, 50, 50);
```

Segeltuch online lesen: https://riptutorial.com/de/javafx/topic/8935/segeltuch

Kapitel 16: Seitennummerierung

Examples

Eine Paginierung erstellen

Seitenumbrüche in JavaFX verwenden einen Rückruf, um die in der Animation verwendeten Seiten abzurufen.

```
Pagination p = new Pagination();
p.setPageFactory(param -> new Button(param.toString()));
```

Dadurch wird eine unendliche Liste von Schaltflächen erstellt, die mit `0...` `setPageFactory` einen Rückruf, der ein int annimmt, und gibt den Knoten zurück, den wir an diesem Index wünschen.

Automatischer Vorlauf

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
fiveSecondsWonder.play();

stage.setScene(new Scene(p));
stage.show();
```

Dadurch wird die Paginierung alle 5 Sekunden vorgerückt.

Wie es funktioniert

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
```

`fiveSecondsWonder` ist eine Zeitleiste, bei der ein Ereignis jedes Mal `fiveSecondsWonder` wird, wenn ein Zyklus beendet wird. In diesem Fall beträgt die Zykluszeit 5 Sekunden.

```
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
```

Kreuze die Paginierung an.

```
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
```

Legen Sie fest, dass die Zeitleiste für immer ausgeführt wird.

```
fiveSecondsWonder.play();
```

Erstellen Sie eine Paginierung von Bildern

```
ArrayList<String> images = new ArrayList<>();
images.add("some\\cool\\image");
images.add("some\\other\\cool\\image");
images.add("some\\cooler\\image");

Pagination p = new Pagination(3);
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Beachten Sie, dass es sich bei den Pfaden um URLs handeln muss, nicht um Dateisystempfade.

Wie es funktioniert

```
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Alles andere ist nur Flaum, hier passiert die eigentliche Arbeit. `setPageFactory` einen Rückruf, der ein int annimmt, und gibt den Knoten zurück, den wir an diesem Index wünschen. Die erste Seite wird dem ersten Element in der Liste zugeordnet, die zweite dem zweiten Element in der Liste und so weiter.

Seitennummerierung online lesen: https://riptutorial.com/de/javafx/topic/5165/seitennummerierung

Kapitel 17: Szenenbildner

Einführung

JavaFX Scene Builder ist ein visuelles Layout-Tool, mit dem Benutzer JavaFX-Anwendungsbenutzeroberflächen schnell und ohne Codierung entwerfen können. Es wird verwendet, um FXML-Dateien zu generieren.

Bemerkungen

JavaFX Scene Builder ist ein visuelles Layout-Tool, mit dem Benutzer JavaFX-Anwendungsbenutzeroberflächen schnell und ohne Codierung entwerfen können. Benutzer können UI-Komponenten per Drag & Drop in einen Arbeitsbereich ziehen, ihre Eigenschaften ändern und Stylesheets anwenden. Der FXML-Code für das von ihnen erstellte Layout wird automatisch im Hintergrund generiert. Das Ergebnis ist eine FXML-Datei, die mit einem Java-Projekt kombiniert werden kann, indem die Benutzeroberfläche an die Anwendungslogik gebunden wird.

Aus der Perspektive eines Model View Controllers (MVC):

- Die FXML-Datei, die die Beschreibung der Benutzeroberfläche enthält, ist die Ansicht.
- Der Controller ist eine Java-Klasse, die optional die Initializable-Klasse implementiert, die als Controller für die FXML-Datei deklariert ist.
- Das Modell besteht aus auf Java-Seite definierten Domänenobjekten, die über den Controller mit der Ansicht verbunden werden können.

Scene Builder Installation

1. Laden Sie die neueste Version von Scene Builder von der Gluon- Website herunter und wählen Sie das Installationsprogramm für Ihre Plattform oder die ausführbare Dose aus.

2. Doppelklicken Sie nach dem Herunterladen des Installationsprogramms, um Scene Builder auf Ihrem System zu installieren. Eine aktualisierte JRE ist enthalten.

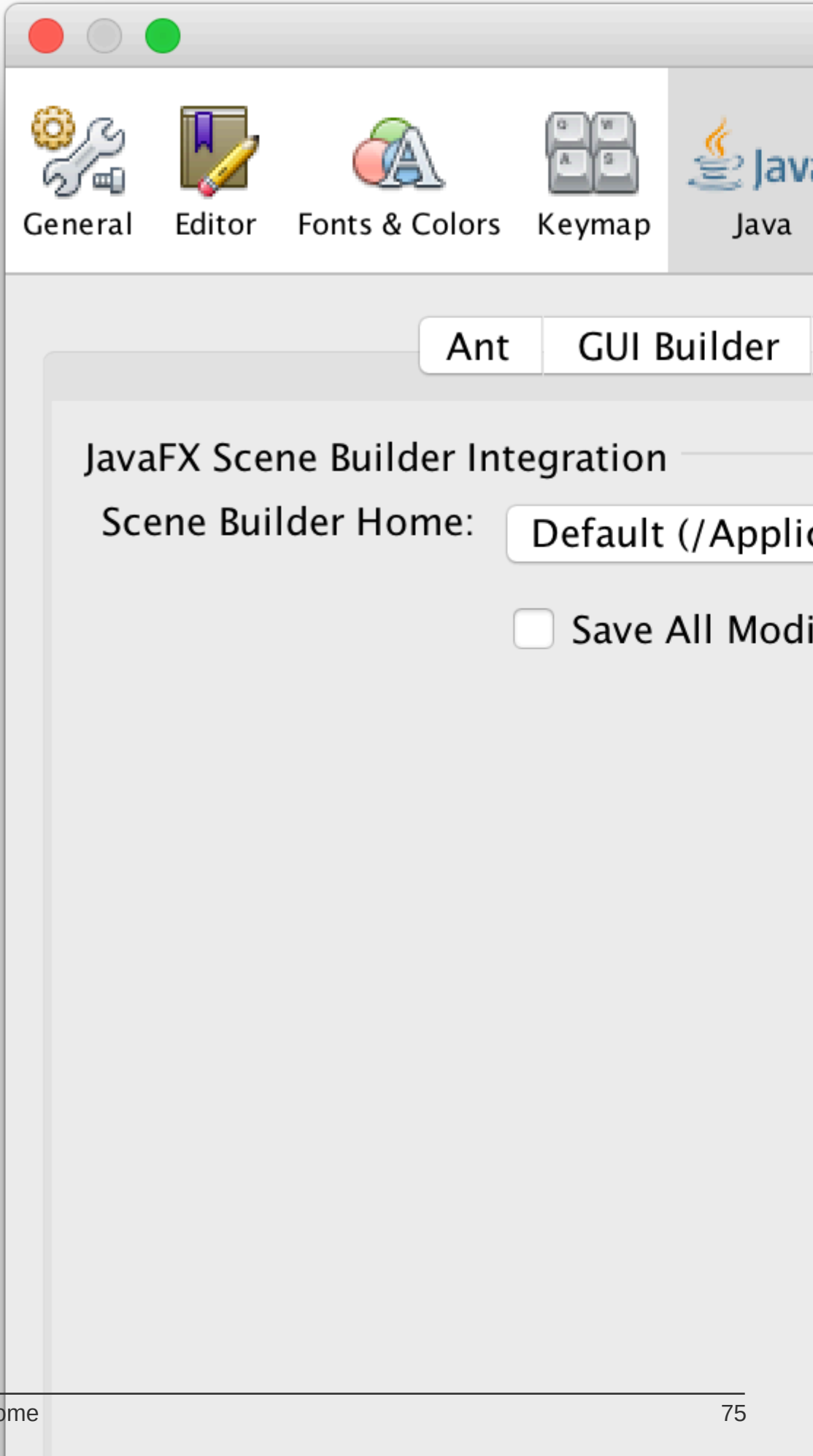
3. Doppelklicken Sie auf das Scene Builder-Symbol, um es als eigenständige Anwendung auszuführen.

4. IDE-Integration

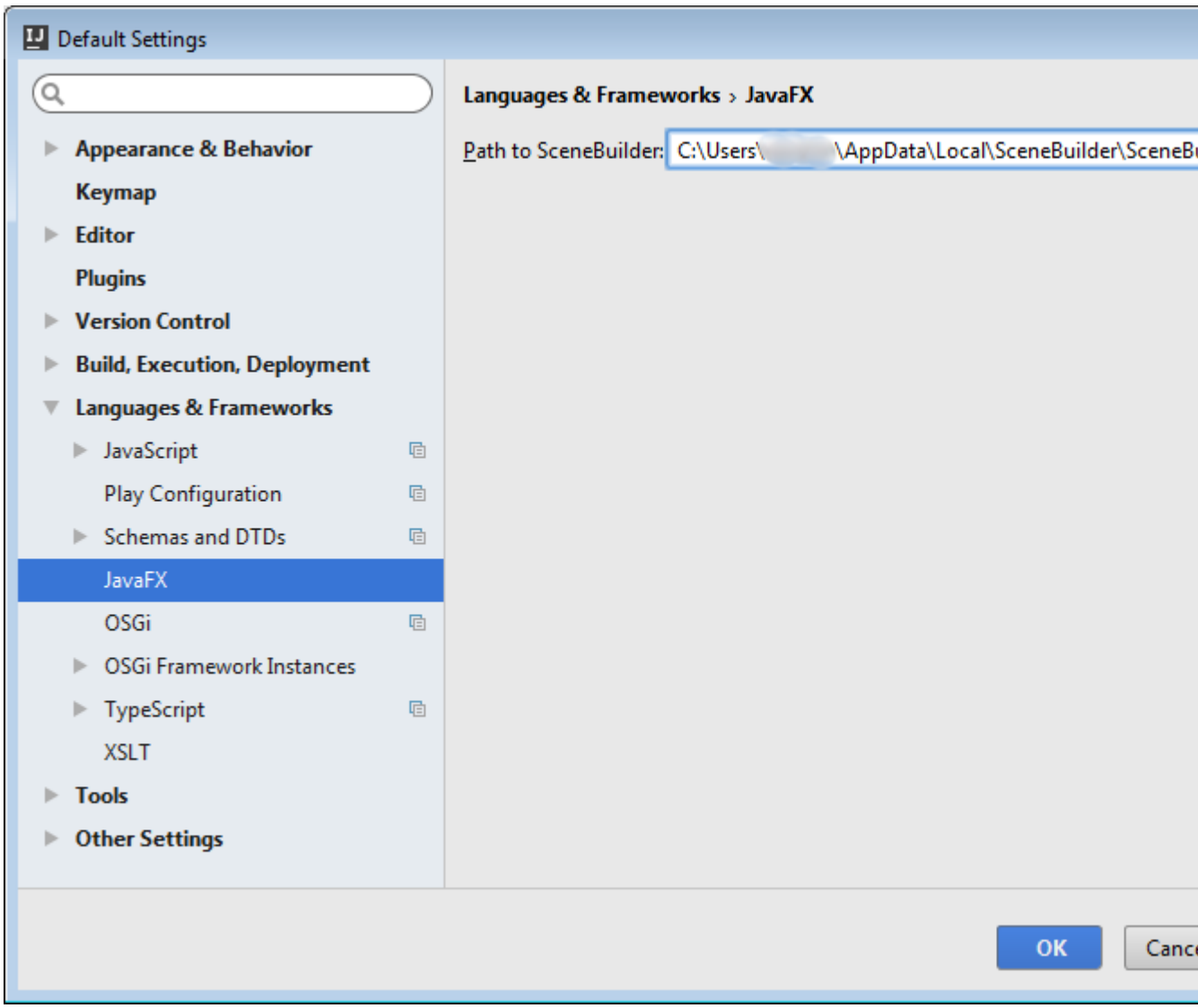
   Obwohl Scene Builder eine eigenständige Anwendung ist, werden FXML-Dateien erstellt, die in ein Java SE-Projekt integriert sind. Beim Erstellen dieses Projekts in einer IDE ist es zweckmäßig, einen Link zum Scene Builder-Pfad anzugeben, damit FXML-Dateien bearbeitet werden können.

   - NetBeans: Unter Windows gehen Sie zu NetBeans-> Tools-> Options-> Java->

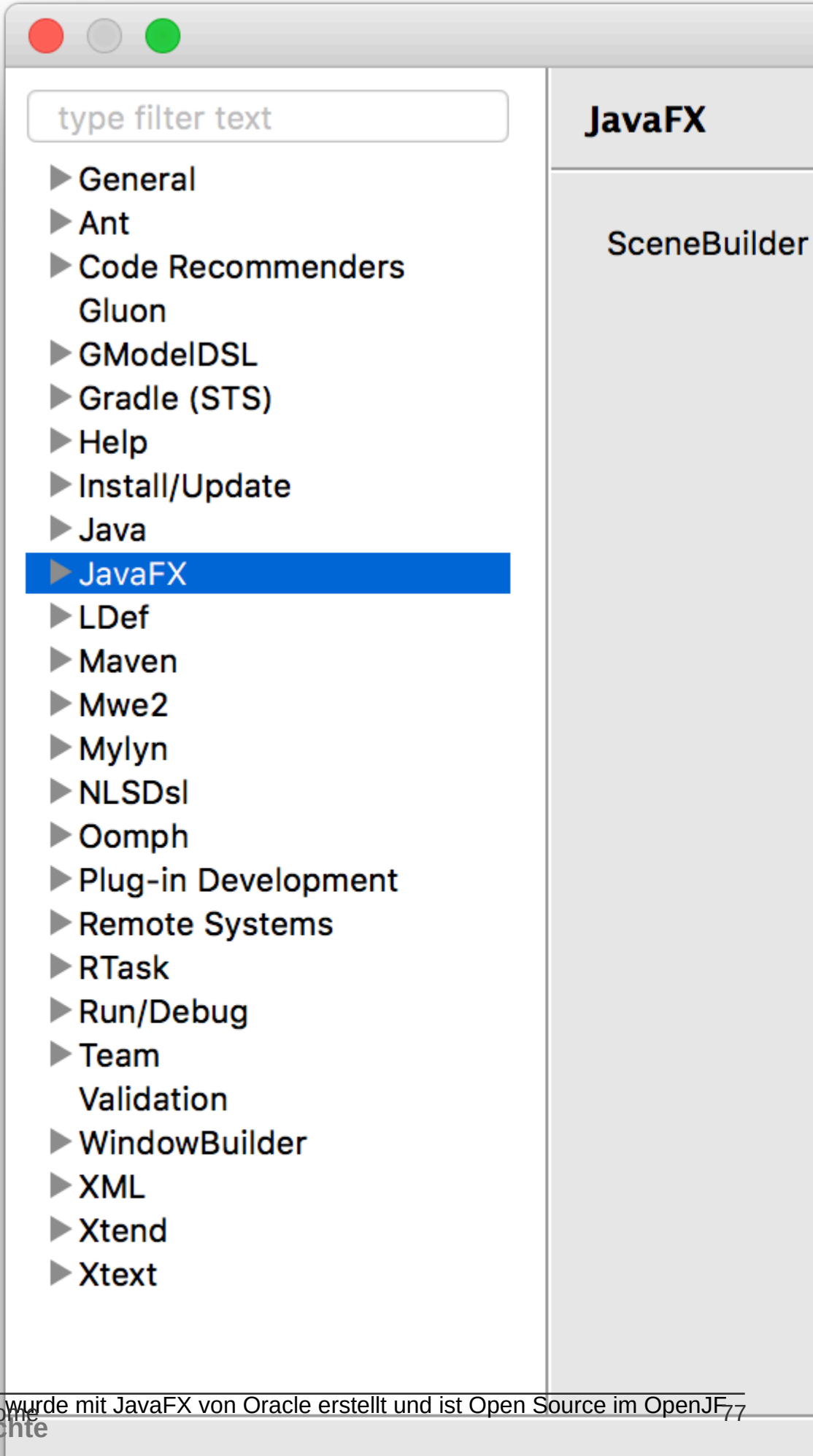
JavaFX. Unter Mac OS X gehen Sie zu NetBeans-> Preferences-> Java-> JavaFX. Geben Sie den Pfad für das Scene Builder-Home an.


General
Editor
Fonts & Colors
Keymap
Java
Ant
GUI Builder
JavaFX Scene Builder Integration
Scene Builder Home:
Default (/Applic
Save All Modi

- IntelliJ: Unter Windows gehen Sie zu IntelliJ-> Einstellungen-> Sprachen & Frameworks> JavaFX. Unter Mac OS X gehen Sie zu IntelliJ -> Einstellungen -> Sprachen und Frameworks -> JavaFX. Geben Sie den Pfad für das Scene Builder-Home an.



- Eclipse: Unter Windows gehen Sie zu Eclipse-> Window-> Preferences-> JavaFX. Unter Mac OS X gehen Sie zu Eclipse-> Preferences-> JavaFX. Geben Sie den Pfad für das Scene Builder-Home an.



Das Scene Builder-Projekt wurde mit JavaFX von Oracle erstellt und ist Open Source im OpenJF

Ein bisschen Geschichte

Oracle stellte

Binaries bis Scene Builder 2.0 zur Verfügung, einschließlich der JavaFX-Funktionen vor der Veröffentlichung von Java SE 8u40. Daher sind neue Funktionen wie die `Spinner` Steuerelemente nicht enthalten.

Gluon übernahm die Distribution der Binärversionen und ein aktueller Scene Builder 8+ kann von hier aus für jede Plattform heruntergeladen werden .

Es enthält die neuesten Änderungen in JavaFX sowie die neuesten Verbesserungen und Fehlerbehebungen.

Das Open Source-Projekt kann hier gefunden werden, wo Probleme, Funktionsanforderungen und Pull-Anforderungen erstellt werden können.

Die Oracle-Legacy-Binärdateien können hier noch heruntergeladen werden .

Tutorials

Scene Builder-Tutorials finden Sie hier:

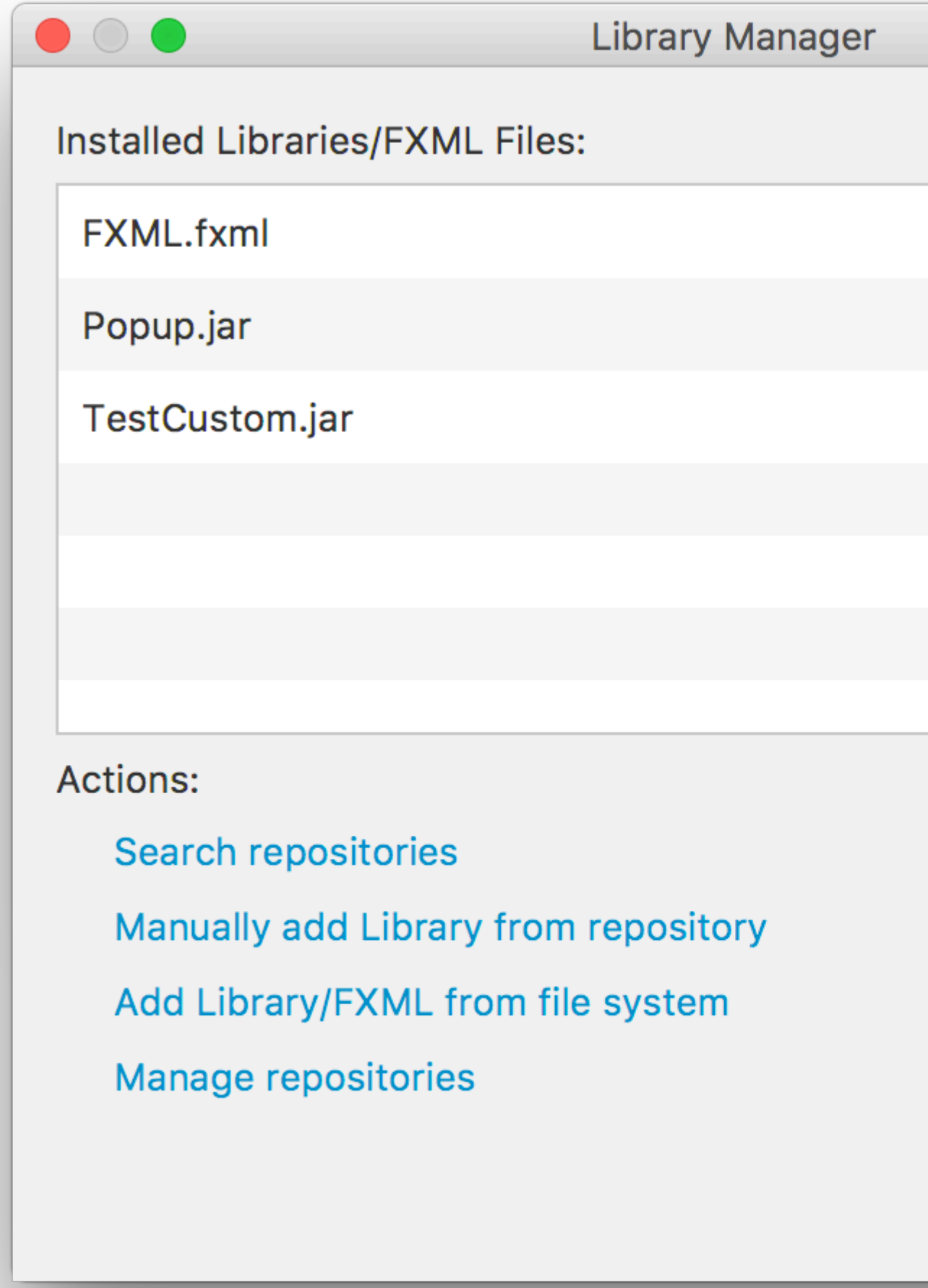
- Oracle Scene Builder 2.0 Tutorial

FXML-Tutorials finden Sie hier.

- Oracle FXML- Tutorial

Benutzerdefinierte Steuerelemente

Gluon hat die neue Funktion, die das Importieren von Drittanbieter-Gläsern mit benutzerdefinierten Steuerelementen ermöglicht, mit dem Bibliotheksmanager (verfügbar seit Scene Builder 8.2.0) vollständig dokumentiert .



Tag bezogen: SceneBuilder

`FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))` , wie von `FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))` .

Beim Laden von `basicFXML.fxml` findet der Loader den Namen der Controller-Klasse, wie in `fx:controller="org.stackoverflow.BasicFXMLController"` in der FXML angegeben.

Dann erstellt der Loader eine Instanz dieser Klasse, in der versucht wird, alle Objekte `@FXML` , die eine `fx:id` in der FXML haben und in der Controller-Klasse mit der `@FXML` FXML-Annotation gekennzeichnet sind.

In diesem Beispiel erstellt der FXMLLoader das Label basierend auf `<Label ... fx:id="label"/>` und `@FXML private Label label;` die `@FXML private Label label;` in das `@FXML private Label label;` des `@FXML private Label label;` .

Wenn die gesamte FXML geladen ist, ruft der FXMLLoader schließlich die `initialize` des Controllers auf, und der Code, der einen Aktionshandler mit der Schaltfläche registriert, wird ausgeführt.

Bearbeitung

Die FXML-Datei kann zwar innerhalb der IDE bearbeitet werden, wird jedoch nicht empfohlen, da die IDE nur eine grundlegende Syntaxprüfung und automatische Vervollständigung, jedoch keine visuelle Anleitung bietet.

Am besten öffnen Sie die FXML-Datei mit Scene Builder, in dem alle Änderungen in der Datei gespeichert werden.

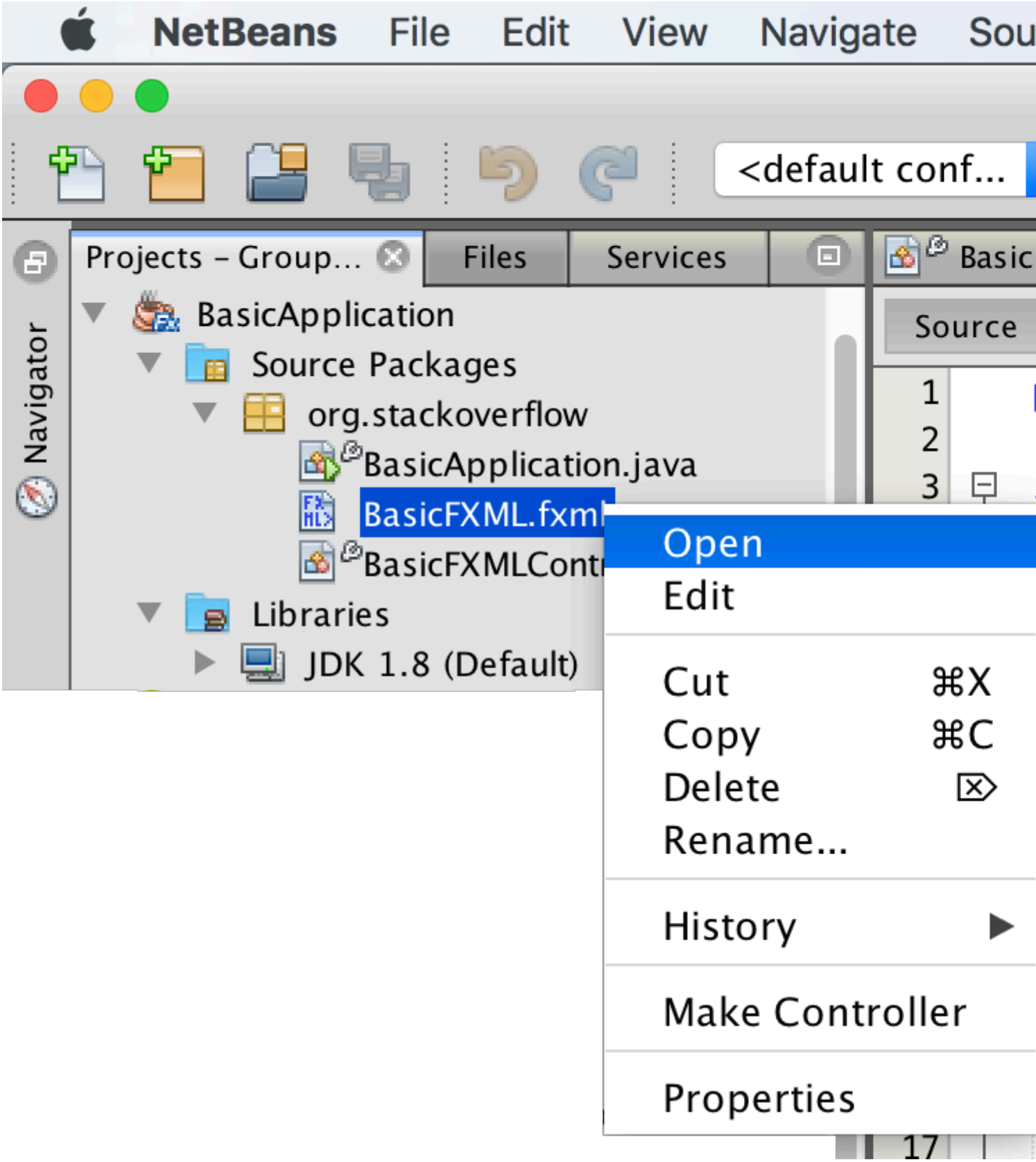
Der Scene Builder kann gestartet werden, um die Datei zu öffnen:

Oder die Datei kann mit Scene Builder direkt von der IDE aus geöffnet werden:

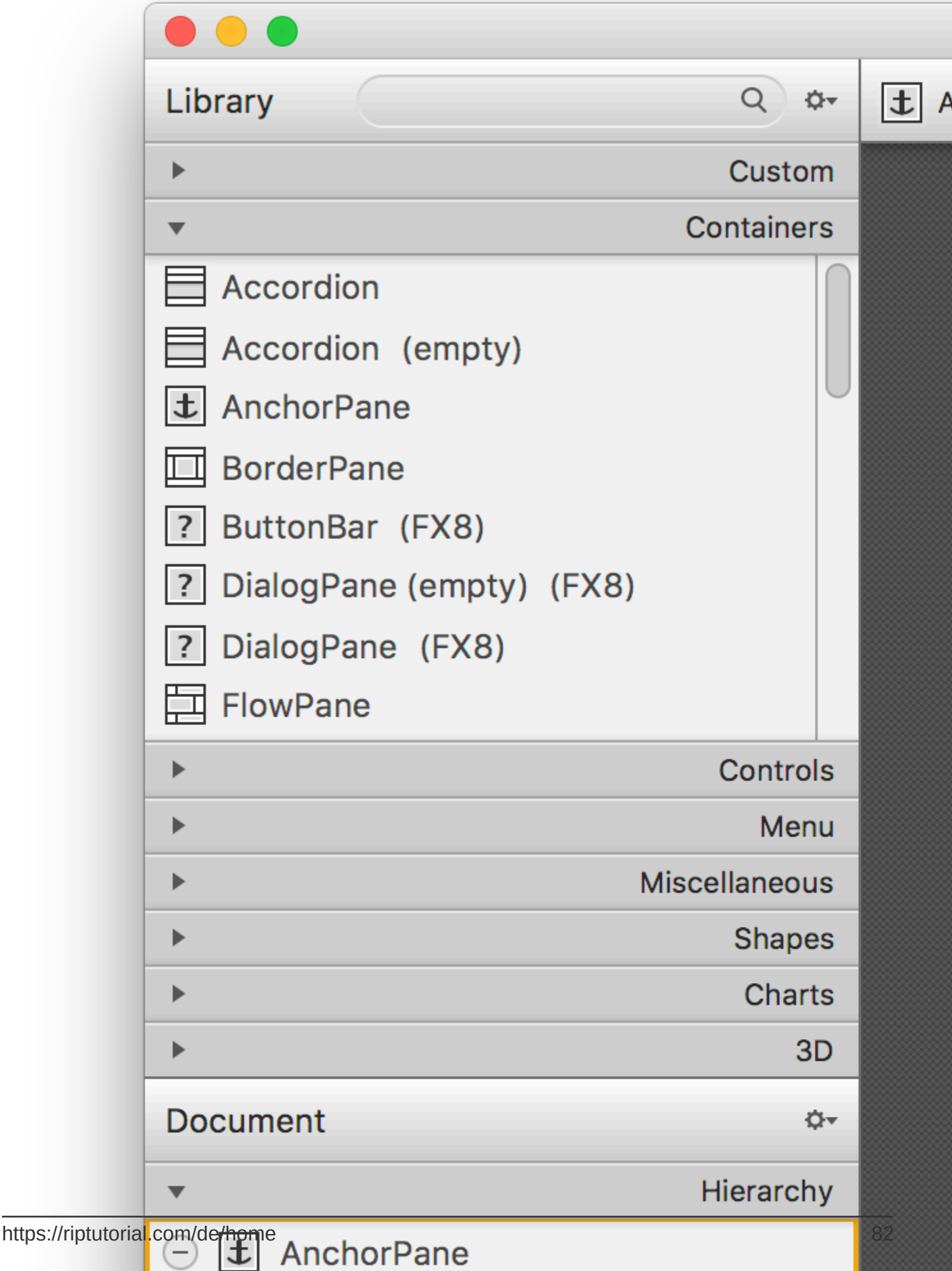
- Doppelklicken Sie in NetBeans auf der Registerkarte "Projekt" auf die Datei oder klicken Sie mit der rechten Maustaste und wählen Sie " `Open` .
- Klicken Sie in IntelliJ auf der Registerkarte "Projekt" mit der rechten Maustaste auf die Datei

und wählen Sie " `Open In Scene Builder` .

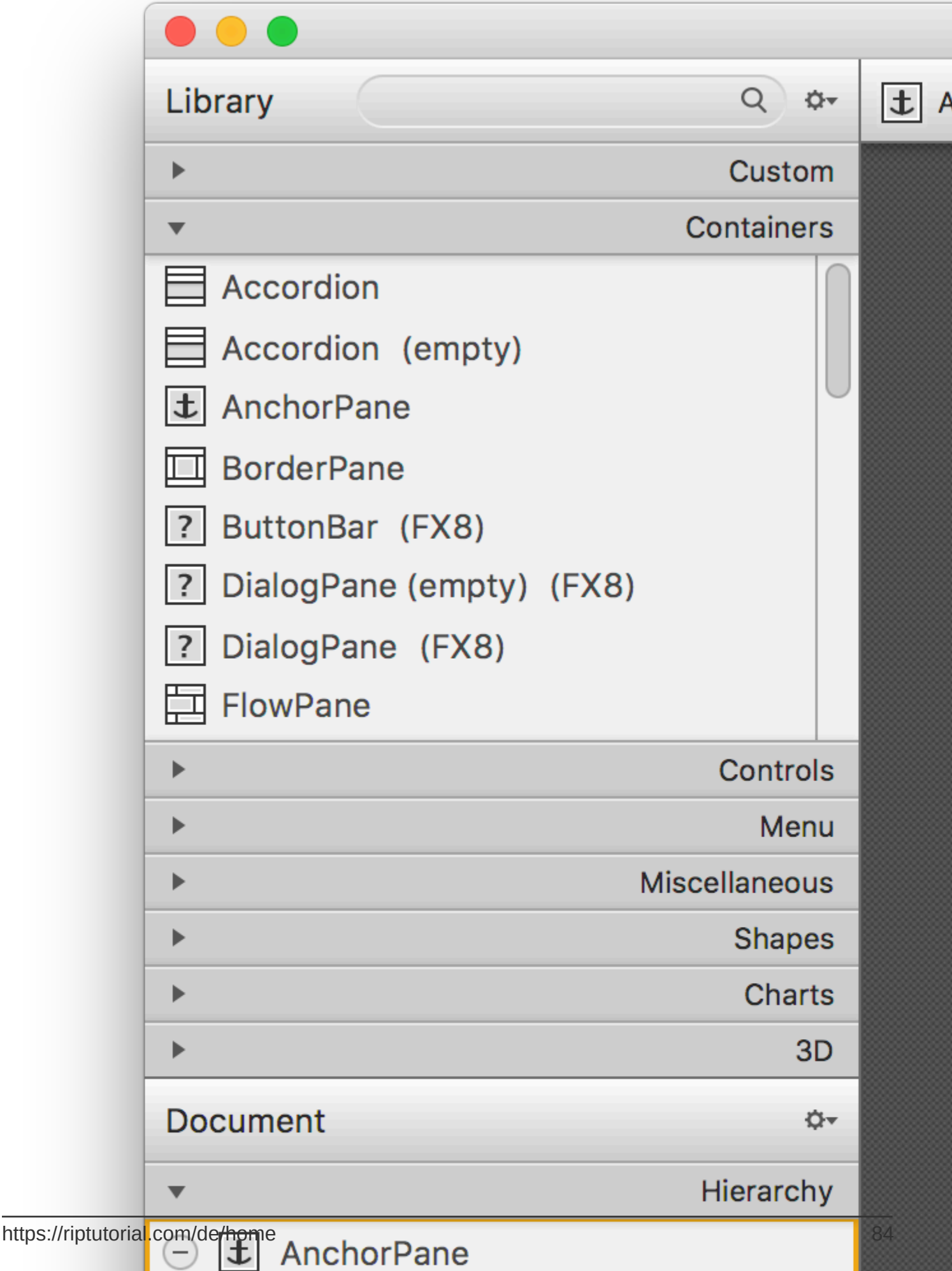
- Klicken Sie in Eclipse auf der Registerkarte "Projekt" mit der rechten Maustaste auf die Datei und wählen Sie " `Open with Scene Builder` .



Wenn Scene Builder ordnungsgemäß installiert und der Pfad zur IDE hinzugefügt wurde (siehe Anmerkungen unten), wird die Datei geöffnet:


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

`fx:id` . Es kann in der festgelegt werden - `Code` - Fenster:


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

Kapitel 18: Tabellenansicht

Examples

Beispiel TableView mit 2 Spalten

Tabellenelement

Die folgende Klasse enthält 2 Eigenschaften, einen Namen (`String`) und die Größe (`double`). Beide Eigenschaften sind in JavaFX-Eigenschaften eingeschlossen, damit `TableView` Änderungen beobachten kann.

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public Person(String name, double size) {
        this.size = new SimpleDoubleProperty(this, "size", size);
        this.name = new SimpleStringProperty(this, "name", name);
    }

    private final StringProperty name;
    private final DoubleProperty size;

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public final double getSize() {
        return this.size.get();
    }

    public final void setSize(double value) {
        this.size.set(value);
    }

    public final DoubleProperty sizeProperty() {
        return this.size;
    }

}
```

Beispielanwendung

Diese Anwendung zeigt eine `TableView` mit 2 Spalten. eine für den Namen und eine für die Größe einer `Person` . Wenn Sie eine der `Person` `TextField` die Daten `TextField` unterhalb der `TableView` und der Benutzer kann die Daten bearbeiten. Beachten Sie, dass die `TableView` automatisch aktualisiert wird, sobald die Bearbeitung `TableView` ist.

Jeder für jede `TableColumn` , die der `TableView` hinzugefügt wird, `TableView` eine `cellValueFactory` zugewiesen. Diese Factory ist für die Konvertierung von Tabellenelementen (`Person`) in `ObservableValue` Werte verantwortlich, die den Wert enthalten, der in der Tabellenzelle angezeigt werden soll. `TableView` kann `TableView` Änderungen für diesen Wert `TableView` .

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.TextFormatter;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;
import javafx.util.StringConverter;

public class TableSample extends Application {

    @Override
    public void start(Stage primaryStage) {
        // data for the tableview. modifying this list automatically updates the tableview
        ObservableList<Person> data = FXCollections.observableArrayList(
                new Person("John Doe", 1.75),
                new Person("Mary Miller", 1.70),
                new Person("Frank Smith", 1.80),
                new Person("Charlotte Hoffman", 1.80)
        );

        TableView<Person> tableView = new TableView<>(data);

        // table column for the name of the person
        TableColumn<Person, String> nameColumn = new TableColumn<>("Name");
        nameColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
String>, ObservableValue<String>>() {

            @Override
            public ObservableValue<String> call(TableColumn.CellDataFeatures<Person, String>
param) {
                return param.getValue().nameProperty();
            }
        });

        // column for the size of the person
        TableColumn<Person, Number> sizeColumn = new TableColumn<>("Size");
```

```
        sizeColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
Number>, ObservableValue<Number>>() {

            @Override
            public ObservableValue<Number> call(TableColumn.CellDataFeatures<Person, Number>
param) {
                return param.getValue().sizeProperty();
            }
        });

        // add columns to tableview
        tableView.getColumns().addAll(nameColumn, sizeColumn);

        TextField name = new TextField();

        TextField size = new TextField();

        // convert input from textfield to double
        TextFormatter<Double> sizeFormatter = new TextFormatter<Double>(new
StringConverter<Double>() {

            @Override
            public String toString(Double object) {
                return object == null ? "" : object.toString();
            }

            @Override
            public Double fromString(String string) {
                if (string == null || string.isEmpty()) {
                    return null;
                } else {
                    try {
                        double val = Double.parseDouble(string);
                        return val < 0 ? null : val;
                    } catch (NumberFormatException ex) {
                        return null;
                    }
                }
            }

        });
        size.setTextFormatter(sizeFormatter);

        Button commit = new Button("Change Item");
        commit.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                Person p = tableView.getSelectionModel().getSelectedItem();
                p.setName(name.getText());
                Double value = sizeFormatter.getValue();
                p.setSize(value == null ? -1d : value);
            }

        });

        // listen for changes in the selection to update the data in the textfields
        tableView.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Person>() {

            @Override
```

```
            public void changed(ObservableValue<? extends Person> observable, Person oldValue,
Person newValue) {
                commit.setDisable(newValue == null);
                if (newValue != null) {
                    sizeFormatter.setValue(newValue.getSize());
                    name.setText(newValue.getName());
                }
            }

        });

        HBox editors = new HBox(5, new Label("Name:"), name, new Label("Size: "), size,
commit);

        VBox root = new VBox(10, tableView, editors);

        Scene scene = new Scene(root);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

PropertyValueFactory

`PropertyValueFactory` kann als `cellValueFactory` in einer `TableColumn` . Es verwendet Reflektion, um auf Methoden zuzugreifen, die einem bestimmten Muster entsprechen, um die Daten von einem `TableView` Element `TableView` :

Beispiel

```
TableColumn<Person, String> nameColumn = ...
PropertyValueFactory<Person, String> valueFactory = new PropertyValueFactory<>("name");
nameColumn.setCellValueFactory(valueFactory);
```

Der Name der Methode, die zum Abrufen der Daten verwendet wird, hängt vom Konstruktorparameter für `PropertyValueFactory` .

- **Eigenschaftsmethode:** Es wird erwartet, dass diese Art von Methode einen `ObservableValue` , der die Daten enthält. Änderungen können beobachtet werden. Sie müssen mit der `<constructor parameter>Property` pattern `<constructor parameter>Property` übereinstimmen und keine Parameter übernehmen.
- **Getter-Methode:** Diese Art von Methode erwartet, dass der Wert direkt zurückgegeben wird (`String` im obigen Beispiel). Der Methodenname muss mit dem Muster übereinstimmen, `get<Constructor parameter>` . Beachten Sie, dass hier `<Constructor parameter>` mit einem *Großbuchstaben* beginnt. Diese Methode sollte keine Parameter annehmen.

Beispielnamen der Methoden

Konstruktorparameter (ohne Anführungszeichen)	Name der Eigenschaftsmethode	Name der Getter-Methode
foo	fooProperty	getFoo
fooBar	fooBarProperty	getFooBar
XYZ	XYZProperty	getXYZ
listIndex	listIndexProperty	getListIndex
ein Wert	aValueProperty	getAValue

Anpassen der TableCell-Ansicht je nach Artikel

Manchmal sollte eine Spalte einen anderen Inhalt als nur den `toString` Wert des `toString` anzeigen. In diesem Fall wird die `TableCell` s durch die erstellte `cellFactory` des `TableColumn` wird angepasst , das Layout auf dem Elemente basiert zu ändern.

Wichtiger Hinweis: `TableView` erstellt nur die `TableCell` , die in der Benutzeroberfläche angezeigt werden. Die Elemente in den Zellen können sich ändern oder sogar leer werden. Der Programmierer muss darauf achten, alle Änderungen an der `TableCell` rückgängig zu machen, die `TableCell` wurden, als ein Element hinzugefügt wurde, wenn es entfernt wurde. Andernfalls wird der Inhalt möglicherweise immer noch in einer Zelle angezeigt, in der "es nicht gehört".

In der folgenden Beispieleinstellung führt ein Element dazu, dass der Text gesetzt wird und das Bild in der `ImageView` angezeigt `ImageView` :

```
image.setImage(item.getEmoji());
setText(item.getValue());
```

Wenn das Element wird `null` oder die Zelle leer wird, werden diese Änderungen rückgängig gemacht , indem die Werte wieder auf Einstellung `null` :

```
setText(null);
image.setImage(null);
```

Das folgende Beispiel zeigt ein Emoji zusätzlich zu Text in einer `TableCell` .

Die `updateItem` Methode wird bei jeder `updateItem` des Elements einer `Cell` aufgerufen. Wenn Sie diese Methode überschreiben, können Sie auf Änderungen reagieren und das Aussehen der Zelle anpassen. Das Hinzufügen eines Listeners zu `itemProperty()` einer Zelle wäre eine Alternative, in vielen Fällen wird `TableCell` jedoch erweitert.

Gegenstandsart

```
import javafx.scene.image.Image;

// enum providing image and text for certain feelings
```

```
public enum Feeling {
    HAPPY("happy",
"https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Emojione_1F600.svg/64px-
Emojione_1F600.svg.png"),
    SAD("sad",
"https://upload.wikimedia.org/wikipedia/commons/thumb/4/42/Emojione_1F62D.svg/64px-
Emojione_1F62D.svg.png")
    ;
    private final Image emoji;
    private final String value;

    Feeling(String value, String url) {
        // load image in background
        emoji = new Image(url, true);
        this.value = value;
    }

    public Image getEmoji() {
        return emoji;
    }

    public String getValue() {
        return value;
    }

}
```

Code in der Anwendungsklasse

```
import javafx.application.Application;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

public class EmotionTable extends Application {

    public static class Item {

        private final ObjectProperty<Feeling> feeling;

        public Item(Feeling feeling) {
            this.feeling = new SimpleObjectProperty<>(feeling);
        }

        public final Feeling getFeeling() {
            return this.feeling.get();
```

```
        }

        public final void setFeeling(Feeling value) {
            this.feeling.set(value);
        }

        public final ObjectProperty<Feeling> feelingProperty() {
            return this.feeling;
        }

    }

    @Override
    public void start(Stage primaryStage) {
        TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD),
                null,
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD)
        ));

        EventHandler<ActionEvent> eventHandler = new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                // change table items depending on userdata of source
                Node source = (Node) event.getSource();
                Feeling targetFeeling = (Feeling) source.getUserData();
                for (Item item : table.getItems()) {
                    if (item != null) {
                        item.setFeeling(targetFeeling);
                    }
                }
            }

        };

        TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

        feelingColumn.setCellValueFactory(new PropertyValueFactory<>("feeling"));

        // use custom tablecell to display emoji image
        feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item,
Feeling>>() {

            @Override
            public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
                return new EmojiCell<>();
            }
        });

        table.getColumns().add(feelingColumn);

        Button sunshine = new Button("sunshine");
        Button rain = new Button("rain");

        sunshine.setOnAction(eventHandler);
```

```
        rain.setOnAction(eventHandler);

        sunshine.setUserData(Feeling.HAPPY);
        rain.setUserData(Feeling.SAD);

        Scene scene = new Scene(new VBox(10, table, new HBox(10, sunshine, rain)));

        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

Zellklasse

```
import javafx.scene.control.TableCell;
import javafx.scene.image.ImageView;

public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {
        // add ImageView as graphic to display it in addition
        // to the text in the cell
        image = new ImageView();
        image.setFitWidth(64);
        image.setFitHeight(64);
        image.setPreserveRatio(true);

        setGraphic(image);
        setMinHeight(70);
    }

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

Schaltfläche zur Tabellenansicht hinzufügen

Sie können Tableview eine Schaltfläche oder eine andere Javafx-Komponente hinzufügen, indem `setCellFactory(Callback value)` Methode `setCellFactory(Callback value)` Spalte verwenden.

Beispielanwendung

In dieser Anwendung fügen wir eine Schaltfläche zu TableView hinzu. Wenn Sie auf diese Spaltenschaltfläche klicken, werden Daten in derselben Zeile wie die Schaltflächen ausgewählt und ihre Informationen gedruckt.

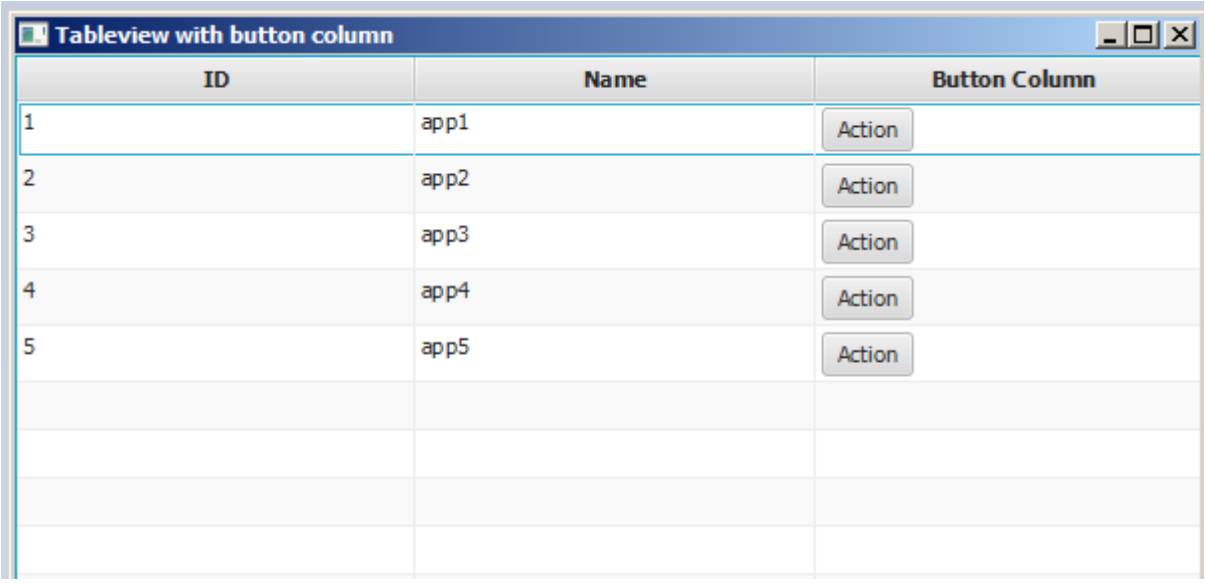
In der `addButtonToTable()` -Methode ist der `cellFactory` Callback dafür verantwortlich, der zugehörigen Spalte eine Schaltfläche hinzuzufügen. Wir definieren die aufrufbare cellFactory und implementieren ihre override-Aufrufmethode `call(...)` , um `TableCell` mit der Schaltfläche und dann dieser `cellFactory` auf die zugehörige Spaltengruppe `setCellFactory(..)` . In unserem Beispiel ist dies `colBtn.setCellFactory(cellFactory)` . SSCCE ist unten:

```
import javafx.application.Application;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import javafx.util.Callback;

public class TableViewSample extends Application {

    private final TableView<Data> table = new TableView<>();
    private final ObservableList<Data> tvObservableList = FXCollections.observableArrayList();

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

    @Override
    public void start(Stage stage) {

        stage.setTitle("Tableview with button column");
        stage.setWidth(600);
        stage.setHeight(600);

        setTableappearance();

        fillTableObservableListWithSampleData();
        table.setItems(tvObservableList);

        TableColumn<Data, Integer> colId = new TableColumn<>("ID");
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Data, String> colName = new TableColumn<>("Name");
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));

        table.getColumns().addAll(colId, colName);

        addButtonToTable();
```

```
        Scene scene = new Scene(new Group(table));

        stage.setScene(scene);
        stage.show();
    }

    private void setTableappearance() {
        table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        table.setPrefWidth(600);
        table.setPrefHeight(600);
    }

    private void fillTableObservableListWithSampleData() {

        tvObservableList.addAll(new Data(1, "app1"),
                                new Data(2, "app2"),
                                new Data(3, "app3"),
                                new Data(4, "app4"),
                                new Data(5, "app5"));
    }

    private void addButtonToTable() {
        TableColumn<Data, Void> colBtn = new TableColumn("Button Column");

        Callback<TableColumn<Data, Void>, TableCell<Data, Void>> cellFactory = new
Callback<TableColumn<Data, Void>, TableCell<Data, Void>>() {
            @Override
            public TableCell<Data, Void> call(final TableColumn<Data, Void> param) {
                final TableCell<Data, Void> cell = new TableCell<Data, Void>() {

                    private final Button btn = new Button("Action");

                    {
                        btn.setOnAction((ActionEvent event) -> {
                            Data data = getTableView().getItems().get(getIndex());
                            System.out.println("selectedData: " + data);
                        });
                    }

                    @Override
                    public void updateItem(Void item, boolean empty) {
                        super.updateItem(item, empty);
                        if (empty) {
                            setGraphic(null);
                        } else {
                            setGraphic(btn);
                        }
                    }
                };
                return cell;
            }
        };

        colBtn.setCellFactory(cellFactory);

        table.getColumns().add(colBtn);

    }

    public class Data {
```

```
        private int id;
        private String name;

        private Data(int id, String name) {
            this.id = id;
            this.name = name;
        }

        public int getId() {
            return id;
        }

        public void setId(int ID) {
            this.id = ID;
        }

        public String getName() {
            return name;
        }

        public void setName(String nme) {
            this.name = nme;
        }

        @Override
        public String toString() {
            return "id: " + id + " - " + "name: " + name;
        }

    }
}
```

Bildschirmfoto:



Tabellenansicht online lesen: https://riptutorial.com/de/javafx/topic/2229/tabellenansicht

Kapitel 19: Taste

Examples

Aktionslistener hinzufügen

Schaltflächen lösen Aktionsereignisse aus, wenn sie aktiviert werden (z. B. ein Klick, eine Tastenkombination für die Schaltfläche wird gedrückt, ...).

```
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

Wenn Sie Java 8+ verwenden, können Sie Lambdas für Aktionslistener verwenden.

```
button.setOnAction((ActionEvent a) -> System.out.println("Hello, World!"));
// or
button.setOnAction(a -> System.out.println("Hello, World!"));
```

Hinzufügen einer Grafik zu einer Schaltfläche

Schaltflächen können eine Grafik haben. `graphic` kann ein beliebiger JavaFX-Knoten sein, wie eine `ProgressBar`

```
button.setGraphic(new ProgressBar(-1));
```

Eine `ImageView`

```
button.setGraphic(new ImageView("images/icon.png"));
```

Oder sogar eine andere Taste

```
button.setGraphic(new Button("Nested button"));
```

Erstellen Sie eine Schaltfläche

Das Erstellen eines `Button` ist einfach:

```
Button sampleButton = new Button();
```

Dadurch wird eine neue `Button` ohne Text oder Grafik erstellt.

Wenn Sie eine erstellen möchten `Button` mit einem Text, verwenden Sie einfach den Konstruktor,

der eine nimmt `String` als Parameter (die die Sets `textProperty` des `Button`):

```
Button sampleButton = new Button("Click Me!");
```

Wenn Sie eine `Button` mit einer Grafik oder einem anderen `Node` erstellen möchten, verwenden Sie diesen Konstruktor:

```
Button sampleButton = new Button("I have an icon", new ImageView(new Image("icon.png")));
```

Standard- und Abbrechen-Schaltflächen

`Button` Schaltflächen-API bietet eine einfache Möglichkeit, Tastenkombinationen allgemeine Tastenkombinationen zuzuweisen, ohne auf die der `Scene` zugewiesene Liste von Beschleunigern zugreifen zu müssen oder explizit die Tastenereignisse zu hören. Es gibt zwei praktische Methoden: `setDefaultButton` und `setCancelButton` :

- Wenn Sie `setDefaultButton` auf `true` wird der `Button` jedes Mal `KeyCode.ENTER` wenn er ein `KeyCode.ENTER` empfängt.
- Wenn `setCancelButton` auf `true` wird `true` wird der `Button` jedes Mal `KeyCode.ESCAPE` wenn er ein `KeyCode.ESCAPE` empfängt.

Im folgenden Beispiel wird eine `Scene` mit zwei Tasten erstellt, die beim Drücken von Eingabe- oder Escape-Tasten ausgelöst werden, unabhängig davon, ob sie scharfgestellt sind oder nicht.

```
FlowPane root = new FlowPane();

Button okButton = new Button("OK");
okButton.setDefaultButton(true);
okButton.setOnAction(e -> {
    System.out.println("OK clicked.");
});

Button cancelButton = new Button("Cancel");
cancelButton.setCancelButton(true);
cancelButton.setOnAction(e -> {
    System.out.println("Cancel clicked.");
});

root.getChildren().addAll(okButton, cancelButton);
Scene scene = new Scene(root);
```

Der obige Code funktioniert nicht, wenn diese `KeyEvents` von einem übergeordneten `Node` :

```
scene.setOnKeyPressed(e -> {
    e.consume();
});
```

Taste online lesen: https://riptutorial.com/de/javafx/topic/5162/taste

Kapitel 20: WebView und WebEngine

Bemerkungen

Das `WebView` ist der JavaFX-Knoten, der in den JavaFX-Komponentenbaum integriert ist. Es verwaltet eine `WebEngine` und zeigt deren Inhalt an.

Die `WebEngine` ist die zugrunde liegende Browser Engine, die im Wesentlichen die gesamte Arbeit erledigt.

Examples

Eine Seite laden

```
WebView wv = new WebView();
WebEngine we = wv.getEngine();
we.load("https://stackoverflow.com");
```

`WebView` ist die UI-Shell um die `WebEngine` herum. Nahezu alle Steuerelemente für die Interaktion mit einer Seite ohne Benutzeroberfläche werden über die `WebEngine` Klasse ausgeführt.

Rufen Sie den Seitenverlauf einer WebView ab

```
WebHistory history = webView.getEngine().getHistory();
```

Die Geschichte ist im Wesentlichen eine Liste von Einträgen. Jeder Eintrag stellt eine besuchte Seite dar und bietet Zugriff auf relevante Seiteninformationen wie URL, Titel und Datum, an dem die Seite zuletzt besucht wurde.

Die Liste kann mit der Methode `getEntries()` abgerufen werden. Der Verlauf und die entsprechende Liste der Einträge ändern sich, wenn `WebEngine` im Web navigiert. Die Liste kann je nach Browseraktionen erweitert oder verkleinert werden. Diese Änderungen können von der ObservableList-API, die die Liste verfügbar macht, überwacht werden.

Der Index des Verlaufseintrags, der der aktuell besuchten Seite zugeordnet ist, wird durch `currentIndexProperty()` . Der aktuelle Index kann verwendet werden, um mithilfe der `go(int)` -Methode zu einem Eintrag im Verlauf zu navigieren. `maxSizeProperty()` legt die maximale Verlaufsgröße fest, `maxSizeProperty()` die Größe der Verlaufsliste

Im Folgenden finden Sie ein Beispiel für das Abrufen und Verarbeiten der Liste der Webprotokollelemente .

Eine `ComboBox` (ComboBox) dient zum Speichern der Verlaufselemente. Durch Verwendung eines `ListChangeListener` für das `WebHistory` die `ComboBox` auf das aktuelle `WebHistory` aktualisiert. In der `ComboBox` befindet sich ein `EventHandler` der auf die ausgewählte Seite umleitet.

```
final WebHistory history = webEngine.getHistory();

comboBox.setItems(history.getEntries());
comboBox.setPrefWidth(60);
comboBox.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent ev) {
        int offset =
                comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    }
});

history.currentIndexProperty().addListener(new ChangeListener<Number>() {

    @Override
    public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number
newValue) {
        // update currently selected combobox item
        comboBox.getSelectionModel().select(newValue.intValue());
    }
});

// set converter for value shown in the combobox:
//   display the urls
comboBox.setConverter(new StringConverter<WebHistory.Entry>() {

    @Override
    public String toString(WebHistory.Entry object) {
        return object == null ? null : object.getUrl();
    }

    @Override
    public WebHistory.Entry fromString(String string) {
        throw new UnsupportedOperationException();
    }
});
```

Senden Sie Javascript-Alarme von der angezeigten Webseite an das Java-Anwendungsprotokoll.

```
private final Logger logger = Logger.getLogger(getClass().getCanonicalName());

WebView webView = new WebView();
webEngine = webView.getEngine();

webEngine.setOnAlert(event -> logger.warning(() -> "JS alert: " + event.getData()));
```

Kommunikation zwischen Java-App und Javascript auf der Webseite

Wenn Sie ein WebView verwenden, um Ihre eigene benutzerdefinierte Webseite anzuzeigen, und diese Webseite Javascript enthält, kann es erforderlich sein, eine bidirektionale Kommunikation zwischen dem Java-Programm und dem Javascript auf der Webseite herzustellen.

Dieses Beispiel zeigt, wie Sie eine solche Kommunikation einrichten.

Die Webseite muss ein Eingabefeld und eine Schaltfläche anzeigen. Wenn Sie auf die Schaltfläche klicken, wird der Wert aus dem Eingabefeld an die Java-Anwendung gesendet, die ihn verarbeitet. Nach der Verarbeitung wird ein Ergebnis an das Javascript gesendet, das wiederum das Ergebnis auf der Webseite anzeigt.

Das grundlegende Prinzip ist, dass für die Kommunikation von Javascript nach Java ein Objekt in Java erstellt wird, das in die Webseite eingefügt wird. Für die andere Richtung wird ein Objekt in Javascript erstellt und von der Webseite extrahiert.

Der folgende Code zeigt den Java-Teil. Ich habe alles in einer Datei gespeichert:

```
package com.sothawo.test;

import javafx.application.Application;
import javafx.concurrent.Worker;
import javafx.scene.Scene;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

import java.io.File;
import java.net.URL;

/**
 * @author P.J. Meisch (pj.meisch@sothawo.com).
 */
public class WebViewApplication extends Application {

    /** for communication to the Javascript engine. */
    private JSObject javascriptConnector;

    /** for communication from the Javascript engine. */
    private JavaConnector javaConnector = new JavaConnector();;

    @Override
    public void start(Stage primaryStage) throws Exception {
        URL url = new File("./js-sample.html").toURI().toURL();

        WebView webView = new WebView();
        final WebEngine webEngine = webView.getEngine();

        // set up the listener
        webEngine.getLoadWorker().stateProperty().addListener((observable, oldValue, newValue)
-> {
            if (Worker.State.SUCCEEDED == newValue) {
                // set an interface object named 'javaConnector' in the web engine's page
                JSObject window = (JSObject) webEngine.executeScript("window");
                window.setMember("javaConnector", javaConnector);

                // get the Javascript connector object.
                javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
            }
        });

        Scene scene = new Scene(webView, 300, 150);
        primaryStage.setScene(scene);
        primaryStage.show();
```

```
        // now load the page
        webEngine.load(url.toString());
    }

    public class JavaConnector {
        /**
         * called when the JS side wants a String to be converted.
         *
         * @param value
         *         the String to convert
         */
        public void toLowerCase(String value) {
            if (null != value) {
                javascriptConnector.call("showResult", value.toLowerCase());
            }
        }
    }
}
```

Wenn die Seite geladen wurde, wird ein `JavaConnector` Objekt (das von der inneren Klasse definiert und als Feld erstellt wird) durch folgende Aufrufe in die Webseite gesetzt:

```
JSObject window = (JSObject) webEngine.executeScript("window");
window.setMember("javaConnector", javaConnector);
```

Das `javascriptConnector` Objekt wird von der Webseite mit abgerufen

```
javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
```

Wenn die `toLowerCase(String)` -Methode aus dem `JavaConnector` aufgerufen wird, wird der übergebene Wert konvertiert und dann über das `javascriptConnector` Objekt zurückgesendet.

Und das ist der HTML- und Javascript-Code:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Sample</title>
    </head>
    <body>
        <main>

            <div><input id="input" type="text"></div>
            <button onclick="sendToJava();">to lower case</button>
            <div id="result"></div>

        </main>

        <script type="text/javascript">
            function sendToJava () {
                var s = document.getElementById('input').value;
                javaConnector.toLowerCase(s);
            };
```

```
            var jsConnector = {
                showResult: function (result) {
                    document.getElementById('result').innerHTML = result;
                }
            };

            function getJsConnector() {
                return jsConnector;
            };
        </script>
    </body>
</html>
```

Die `sendToJava` Funktion ruft die Methode des `JavaConnector` die durch den Java-Code festgelegt wurde:

```
function sendToJava () {
    var s = document.getElementById('input').value;
    javaConnector.toLowerCase(s);
};
```

und die Funktion, die vom Java-Code aufgerufen wird, um den `javascriptConnector` abzurufen, gibt nur das `jsConnector` Objekt zurück:

```
var jsConnector = {
    showResult: function (result) {
        document.getElementById('result').innerHTML = result;
    }
};

function getJsConnector() {
    return jsConnector;
};
```

Der Argumenttyp der Aufrufe zwischen Java und Javascript ist nicht auf Strings beschränkt. Weitere Informationen zu den möglichen Typen und Konvertierungen finden Sie im JSObject-API-Dokument .

WebView und WebEngine online lesen: https://riptutorial.com/de/javafx/topic/5156/webview-und-webengine

Kapitel 21: Windows

Examples

Ein neues Fenster erstellen

Um einige Inhalte in einem neuen Fenster anzuzeigen, muss eine `Stage` erstellt werden. Nach der Erstellung und Initialisierung muss `show` oder `showAndWait` für das `Stage` Objekt aufgerufen werden:

```
// create sample content
Rectangle rect = new Rectangle(100, 100, 200, 300);
Pane root = new Pane(rect);
root.setPrefSize(500, 500);

Parent content = root;

// create scene containing the content
Scene scene = new Scene(content);

Stage window = new Stage();
window.setScene(scene);

// make window visible
window.show();
```

Anmerkung: Dieser Code muss auf dem JavaFX-Anwendungsthread ausgeführt werden.

Benutzerdefiniertes Dialogfeld erstellen

Sie können benutzerdefinierte Dialoge erstellen, die viele Komponenten enthalten, und viele Funktionen dafür ausführen. Es verhält sich wie die zweite Stufe auf der Eigentümerstufe.
Im folgenden Beispiel wird eine Anwendung erstellt, die Person in der Tabellensicht der Hauptphase zeigt und eine Person in einem Dialog (AddingPersonDialog) erstellt. GUIs, die mit SceneBuilder erstellt wurden, können jedoch mit reinem Java-Code erstellt werden.

Beispielanwendung:

AppMain.java

```
package customdialog;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class AppMain extends Application {

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

```
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
    @FXML
    private TableColumn colName;
    @FXML
    private TableColumn colAge;

    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
    }

    @Override
```

```
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
               </columnResizePolicy>
            </TableView>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {

    @FXML
    private TextField tfId;
```

```
    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

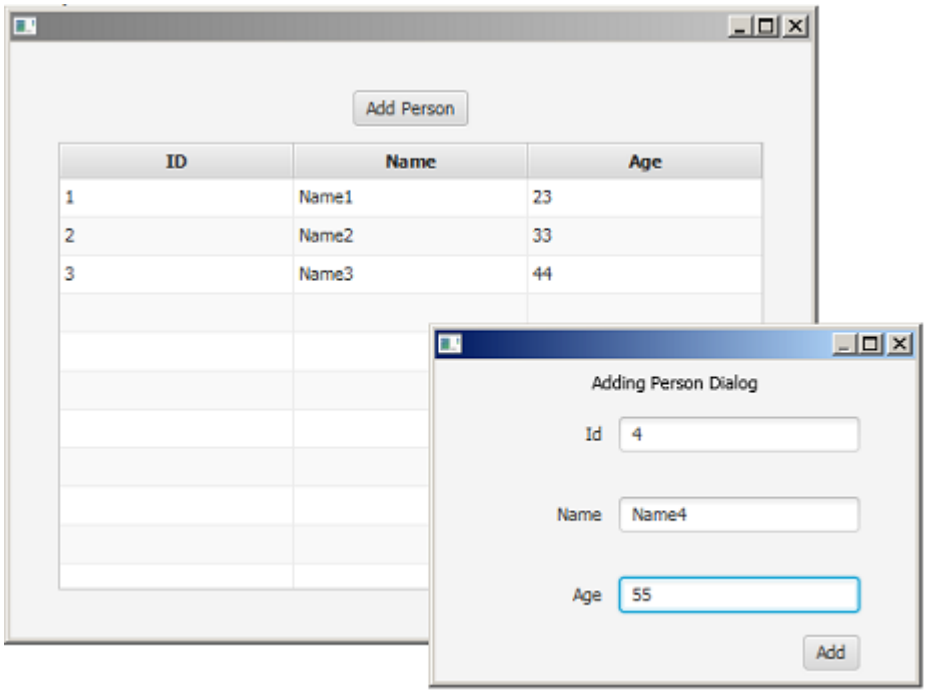
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
```

```
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }

    public int getId() {
        return id.get();
```

```
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```



Bildschirmfoto

Benutzerdefiniertes Dialogfeld erstellen

Sie können benutzerdefinierte Dialoge erstellen, die viele Komponenten enthalten, und viele Funktionen dafür ausführen. Es verhält sich wie die zweite Stufe auf der Eigentümerstufe.
Im folgenden Beispiel wird eine Anwendung erstellt, die Person in der Tabellensicht der Hauptphase zeigt und eine Person in einem Dialog (AddingPersonDialog) erstellt. GUIs, die mit

SceneBuilder erstellt wurden, können jedoch mit reinem Java-Code erstellt werden.

Beispielanwendung:

AppMain.java

```
package customdialog;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class AppMain extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
    @FXML
    private TableColumn colName;
    @FXML
    private TableColumn colAge;
```

```
    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
               </columnResizePolicy>
            </TableView>
         </children>
      </VBox>
   </children>
```

```
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {

    @FXML
    private TextField tfId;

    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

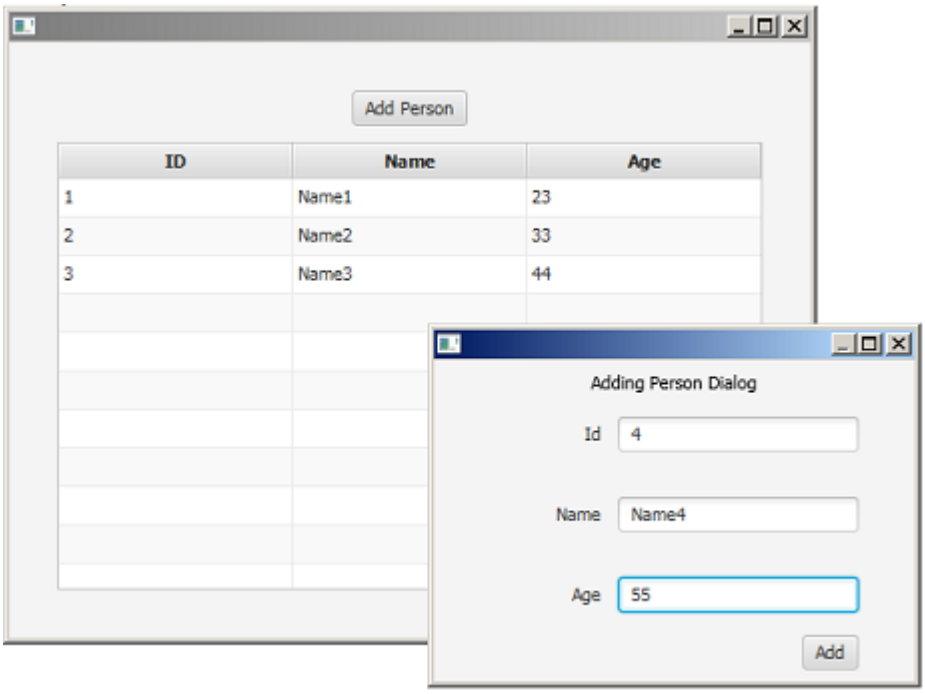
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
```

```
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }

    public int getId() {
        return id.get();
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```

Bildschirmfoto



Windows online lesen: https://riptutorial.com/de/javafx/topic/1496/windows

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Javafx	Community, CraftedCart, D3181, DVarga, fabian, Ganesh, Hendrik Ebbers, Petter Friberg
2	Animation	fabian, J Atkin
3	CSS	fabian
4	Diagramm	Dth, James_D, Jinu P C
5	Dialoge	fabian, GltknBtn, Modus Tollens
6	Drucken	J Atkin, Squidward
7	Eigenschaften & beobachtbar	fabian
8	Einfädeln	Brendan, fabian, GOXR3PLUS, James_D, Koko Essam, sazzy4o
9	FXML und Controller	D3181, fabian, James_D
10	Internationalisierung in JavaFX	ItachiUchiha, Joffrey, Nico T, P.J.Meisch
11	JavaFX-Bindungen	Alexiy
12	Layouts	DVarga, fabian, Filip Smola, Jinu P C, Sohan Chowdhury, trashgod
13	Radio knopf	Nico T
14	ScrollPane	Bo Halim
15	Segeltuch	Dth
16	Seitennummerierung	fabian, J Atkin
17	Szenenbildner	Ashlyn Campbell, José Pereda
18	Tabellenansicht	Bo Halim, fabian, GltknBtn
19	Taste	Dth, DVarga, J Atkin, Maverick283, Nico T, Squidward
20	WebView und	fabian, J Atkin, James_D, P.J.Meisch, Squidward

	WebEngine	
21	Windows	fabian, GltknBtn