LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Fortgeschrittene

Einheit 05: FXML & Scene Builder

Kyrill Schmid
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

FXML und Scene Builder

- Scene Builder installieren und Demo
- Controller einbinden

Praxis:

- Chat-Fenster mit Scene Builder

Lernziele

- Scene Builder und FXML in JavaFX nutzen können

Motivation: FXML

JavaFX bereits einfacher zu implementieren als Swing:

- Einfachere UI-Elemente
- Trennung vom Layout durch Laden von externen CSS-Files

Dennoch:

- GUI Programmierung immer noch aufwendig
- Und eigentlich trivial
  - Folgt immer dem selben Muster

Daher:

- Einführung eines XML-Standards: FXML
  - Formale Beschreibung von Layout und UI-Elementen
  - Können dann als Objekte geladen und im Code verwendet werden
- Vorteil: Einfache Erzeugung durch WYSIWYG-Tools
  - Hier bspw.: Scene Builder

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

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

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xml
   <center>
      <GridPane prefHeight="338.0" prefWidth="600.0" BorderPane.alignment="CENTER">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="294.0" minWidth="10.0" prefWidth="146.0" />
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="532.0" minWidth="10.0" prefWidth="454.0" />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints maxHeight="128.0" minHeight="0.0" prefHeight="12.0" vgrow="SOMETIMES" />
          <RowConstraints maxHeight="319.0" minHeight="10.0" prefHeight="307.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
           <Label text="Counter" />
           <Text fx:id="actionTarget" strokeType="OUTSIDE" strokeWidth="0.0" text="0.0" GridPane.columnIndex="1" />
           <HBox alignment="TOP_CENTER" prefHeight="100.0" prefWidth="200.0" spacing="10.0" GridPane.rowIndex="1">
              <children>
                 <Button mnemonicParsing="false" onAction="#handleIncreaseButton" text="Increase" />
                 <Button mnemonicParsing="false" onAction="#handleDecreaseButton" prefHeight="25.0" prefWidth="66.0" text="Decrease" />
              </children>
              <opaqueInsets>
                 <Insets top="4.0" />
              </opaqueInsets>
           </HBox>
         </children>
      </GridPane>
   </center>
</BorderPane>
```

Beispiel für eine FXML-Datei


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

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


<AnchorPane prefHeight="219.0" prefWidth="213.0"
xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8">
   <children>
      <TitledPane animated="false" layoutX="-5.0" prefHeight="247.0" prefWidth="221.0" text="How is
         it?">
        <content>
          <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="215.0" prefWidth="200.0">
               <children>
                  <VBox layoutX="53.0" layoutY="11.0" prefHeight="200.0" prefWidth="100.0">
                     <children>
                        <ImageView fitHeight="175.0" fitWidth="83.0" pickOnBounds="true"
                          preserveRatio="true">
                            <image>
                               <Image url="@../../../../Pictures/2000px-Wave.svg.png" />
                            </image>
                        </ImageView>
                        <TextField text="How is it" />
                     </children>
                  </VBox>
               </children>
            </AnchorPane>
        </content>
      </TitledPane>
   </children>
</AnchorPane>
```

„Normaler“ XML-Header

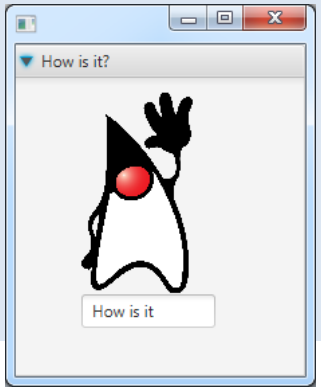
Imports: Wichtig für das Laden der Objekte

Root Node

Definition des XML-Namespaces für fxml und javafx

Kind Knoten

Klassische XML Baumstruktur: Tags schließen!



FXML Datei erzeugen: Scene Builder

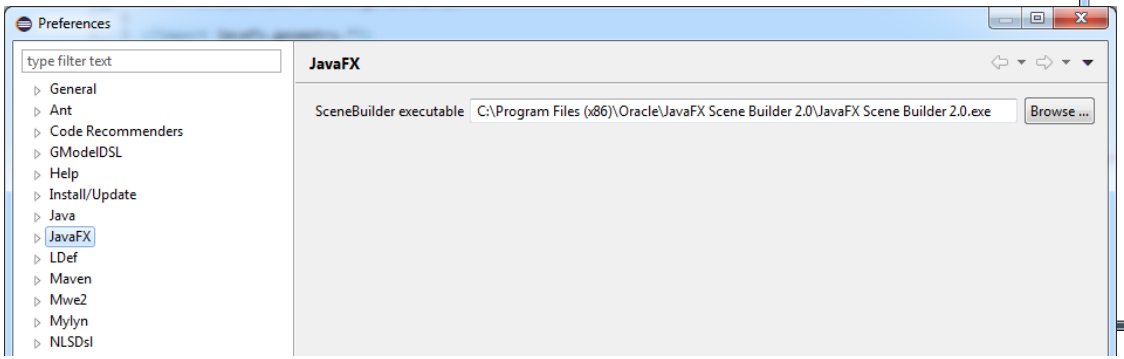
I.d.R. schreibt man eine solche FXML-Datei nicht selbst!

- Wäre sehr umständlich
- Sondern, man lässt sie erzeugen mittels WYSIWYG-Tool: **Scene Builder**

Download bei Oracle unter:

http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html

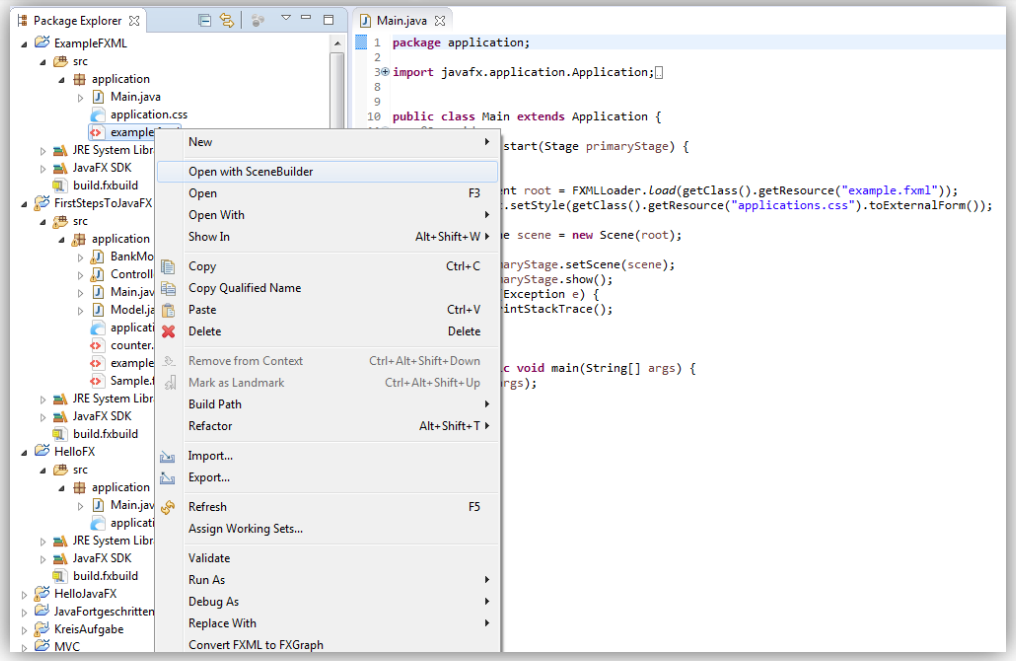
- Installieren und einbinden bei Eclipse:
  - *Window->Preferences->JavaFX:*



Scene Builder: Demo

Neues JavaFX Projekt in Eclipse:

- Wir ändern den vorgeschlagenen Code um die Root Pane
  - FXML-Datei „example.fxml“
- Neue FXML Datei mit entspr. Namen anlegen
  - Mit Scene-Builder gestalten




```
// Im Code FXML-Datei laden:
Parent root = FXMLLoader.load(getClass().getResource("example.fxml"));

// Szene setzen und CSS Datei anbinden:
Scene scene = new Scene(root);
scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

// Szene auf die Bühne holen und anzeigen:
primaryStage.setScene(scene);
primaryStage.show();
```

Einen Controller bauen

Jede View (FXML-File) braucht einen Controller

- Den können wir durch eine separate Klasse spezifizieren:
  - Sollte das Interface `Initializable` implementieren
    - Hat die Methode `void initialize(URL location, ResourceBundle resources)`
    - Wird aufgerufen, wenn der Inhalt des FXML-Dokuments komplett geladen wurde
    - Zum Vergleich: Der Konstruktor wird zuerst aufgerufen (vor dem Laden der UI Elemente)

```
// Beispiel: Leerer Controller

public class Controller implements Initializable{

   public Controller(){

      System.out.println("Klasse instanziiert");
   }


   @Override
   public void initialize(URL location, ResourceBundle resources) {

      System.out.println(“Elemente Geladen");

   }
}
```

Ausgabe:
```
Klasse instanziiert
Elemente Geladen
```

Den Controller hinzufügen

Den Controller können wir nun sehr einfach einbinden und in Ihm die Nutzereingaben entgegennehmen:

```
… xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.Controller">
```

- Verbindung zwischen Controller und UI Elementen über IDs und FXML Annotations: @FXML

```
@FXML private Text actionTarget;
```

```
<Text fx:id="actionTarget"
strokeType="OUTSIDE"
strokeWidth="0.0" text="0.0"
GridPane.columnIndex="1" />
```

Im Controller | Im FXML File

- Verbindung zwischen EventHandler und ActionEvent über onAction und FXML Annotation:

```
@FXML protected void handleButton(){
    // Do something…
}
```

```
<Button
onAction="#handleButton"
text="Increase" />
```

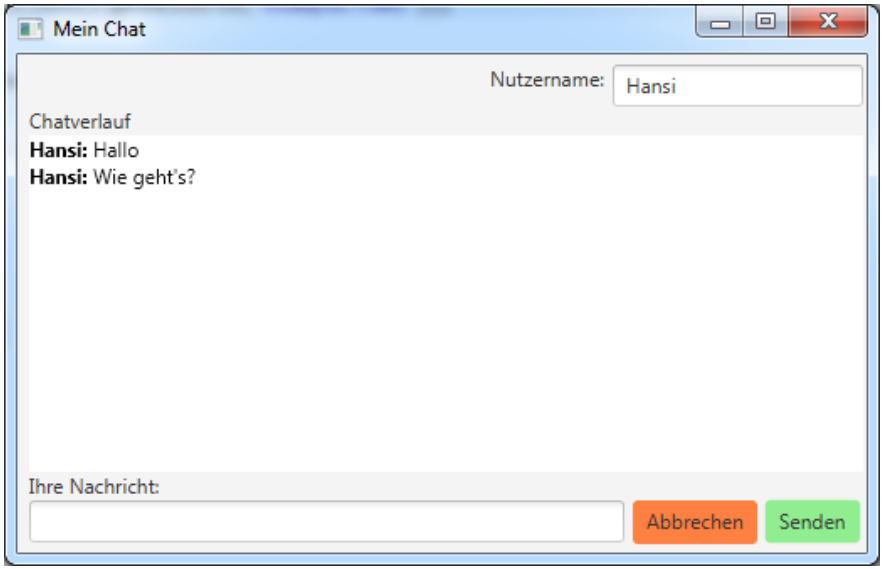
Im Controller | Im FXML File

Beispiel für einen Controller


```
package application;

import java.net.URL;
import java.util.ResourceBundle;

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

public class Controller implements Initializable{

    private Model model;

    public Controller(){

        System.out.println("Klasse instanziiert");
    }

    @FXML private Text actionTarget;

    @FXML protected void handleIncreaseButton(){

        model.setCounter(model.getCounter()+1);
        actionTarget.setText(""+model.getCounter());
    }

    @FXML protected void handleDecreaseButton(){
        model.setCounter(model.getCounter()-1);
        actionTarget.setText(""+model.getCounter());
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {

        System.out.println("Geladen");
        this.model = new Model();
    }
}
```

Aufgabe zu FXML mit Scene Builder

Nutzen Sie nun Ihr erlangtes Wissen über FXML und Scene Builder und gestalten Sie damit eine Chat-GUI, die ungefähr so aussehen könnte:



Schreiben Sie einen geeigneten Controller, der

- Nutzereingaben für Nachrichten und Nutzernamen entgegennimmt
- Beim Drücken des Senden-Buttons diese entsprechend der Grafik im Verlaufsfenster anzeigt.
- Beim Drücken des Abbrechen-Buttons den eingetragenen Text im Nachrichtenfeld wieder löscht

Achten Sie auf eine herkömmliche Usability, d.h.:

- Das Chat-Verlaufsfenster darf nicht direkt editiert werden können
- Beim Absenden einer Nachricht wird automatisch das Nachrichtenfeld geleert.

Ein Model für verschiedene Controller

Will man sein Model für verschiedene Controller nutzen, muss man die Instanz dem Controller bekanntmachen (übergeben).

- Ein einfache Parameterübergabe mittels Konstruktor funktioniert hier nicht, da der Controller mittels FXML vom Framework erzeugt wird!
- Daher 2 Möglichkeiten:

1. Möglichkeit:
   - Die Controller-Instanz selbst erzeugen und nicht durch FXML
     - Ganz normal mittels `new Controller (param1, param2,…)`
   - und beim FXML-Loader bekannt machen:
     - `setController(myController);`

```
Model model = new Model();
FXMLLoader loader = new FXMLLoader(getClass().getResource("counter.fxml"));

Controller c = new Controller(model);
loader.setController(c);

Parent root = loader.load();
//…
```

Der Controller brauch natürlich einen entspr. Konstruktor!

Nicht vergessen: Im FXML File den Controller zu löschen!

Dann erst laden!

Ein Model für verschiedenen Controller

2. Möglichkeit:
   - Den Controller durch FXML erzeugen lassen
   - Nachdem der Controller geladen ist
     - die entsprechende Controller-Instanz durch getController() vom loader holen
     - Einen eigenen Setter zum Setzen des Models aufrufen

```
Model m = new Model();
FXMLLoader loader = new FXMLLoader(getClass().getResource("counter.fxml"));

// Zuerst laden:
Parent root = loader.load();

// Dann Controller-Instanz holen:
Controller c = loader.<Controller>getController();

// Dann eigenen Setter aufrufen:
c.setModel(m);
```

Chat-Fenster mit Logik erweitern

Schreiben Sie nun ein Model für Ihre zuvor erstellte Chat-Anwendung, welches die eingegebenen Nachrichten zusammen mit dem Nutzernamen speichert.

Verknüpfen Sie anschließend das Model mit 2 Chat-Anzeigen, so dass über dieses Model ein lokaler (einfacher) Chat realisiert werden kann.

Ihr Model soll ganz einfach gehalten sein:

- Es besitzt 2 ArrayLists für Nutzernamen und Nachrichten
- Es hat eine Methode, um neue Nachrichten + Nutzernamen zu speichern
  - Bsp.: `public void addMsg(String uname, String txt)`
- Es hat 2 Getter für die beiden ArrayLists

Wenn Sie nun eine Nachricht in das Chat-Fenster eingeben, soll diese Nachricht mit dem Nutzernamen im Model gespeichert werden.

- Für die Ausgabe müssen Sie nun auf den Inhalt der beiden ArrayLists im Model zugreifen und diese wie gewohnt anzeigen.

Problem

Wir haben wieder das Problem von inkonsistenten Anzeigen, obwohl das Model immer die richtigen Datenbestände hat.

- Wir brauchen wieder einen Mechanismus, wie das Observer-Pattern
- Dieses mal: **Properties & Binding**!