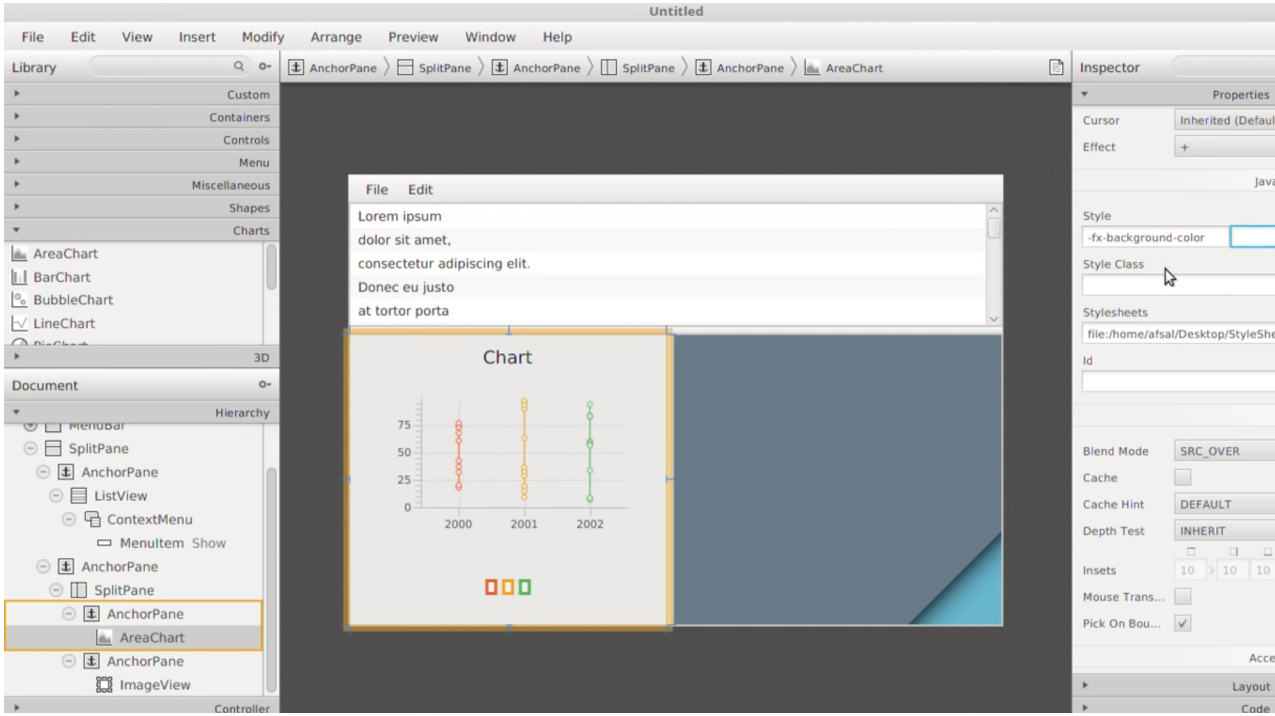
Praktikum „SEP: Java-Programmierung“ WS 2018/19

Einführung JavaFX (OpenJFX)

Thomas Lemberger und Martin Spießl

Überblick



- Plattformübergreifende GUIs für Java
- JavaFX Teil von Java 8
- Für Java 11:
  Separat als OpenJFX
  `openjfx.io/openjfx-docs`
- Tutorial:
  `tinyurl.com/Java8FXTutorial`

Installation unter Java 11

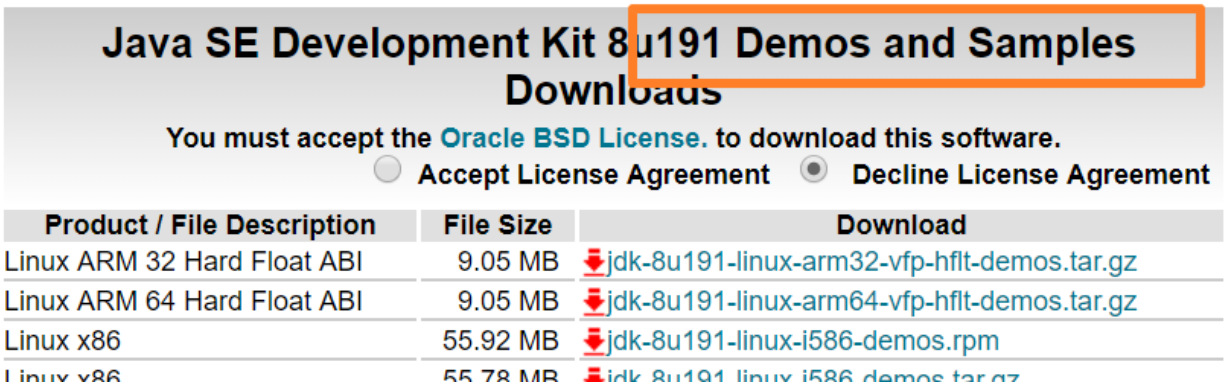
- Anleitung:
  `openjfx.io/openjfx-docs/#install-javafx`

Fähigkeiten

- Web-View
- Interoperabilität mit Swing
- Darstellung von Mediendateien (z.B. Audio, Video)
- 3D-Rendering
- Hardwarebeschleunigung
- Multi-touch und HiDPI support

Beispiele

- Absolute Empfehlung: Beispielprogramm Ensemble
  tinyurl.com/FXSample



Java SE Development Kit 8u191 Demos and Samples Downloads

You must accept the Oracle BSD License. to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	9.05 MB	jdk-8u191-linux-arm32-vfp-hflt-demos.tar.gz
Linux ARM 64 Hard Float ABI	9.05 MB	jdk-8u191-linux-arm64-vfp-hflt-demos.tar.gz
Linux x86	55.92 MB	jdk-8u191-linux-i586-demos.rpm
Linux x86	55.78 MB	jdk-8u191-linux-i586-demos.tar.gz

- Zeigt über 100 Beispielprogramme mit Source Code, um alle Möglichkeiten von JavaFX zu demonstrieren

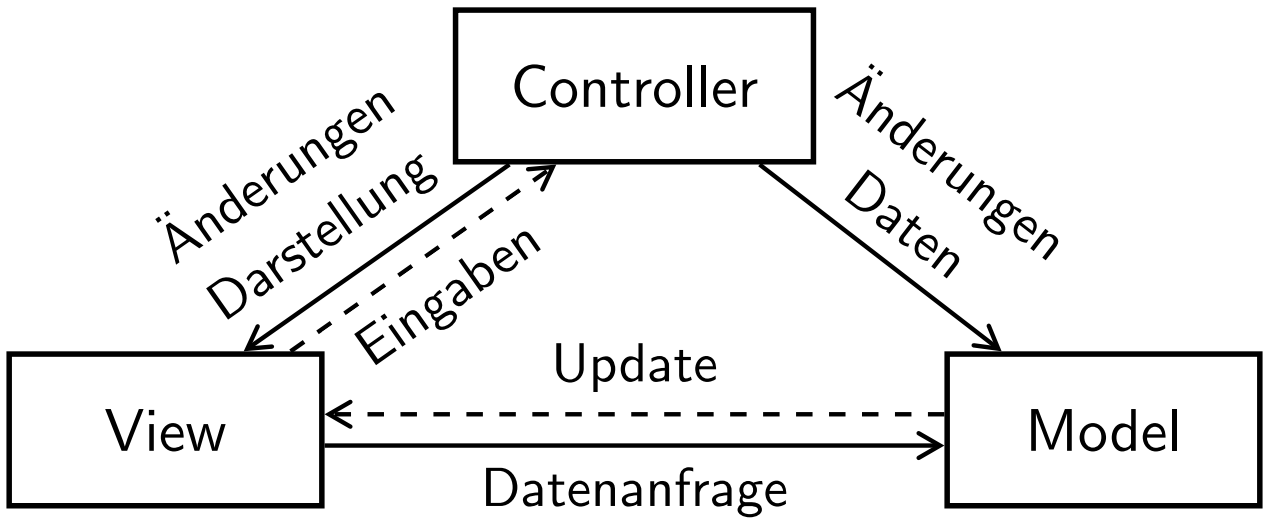
  Datei im entpackten Zip: `demo/javafx_samples/Ensemble8.jar`

  Sourcecode aller Beispiele:

  `demo/javafx_samples/src/Ensemble8/src/samples/java/ensemble/samples/`

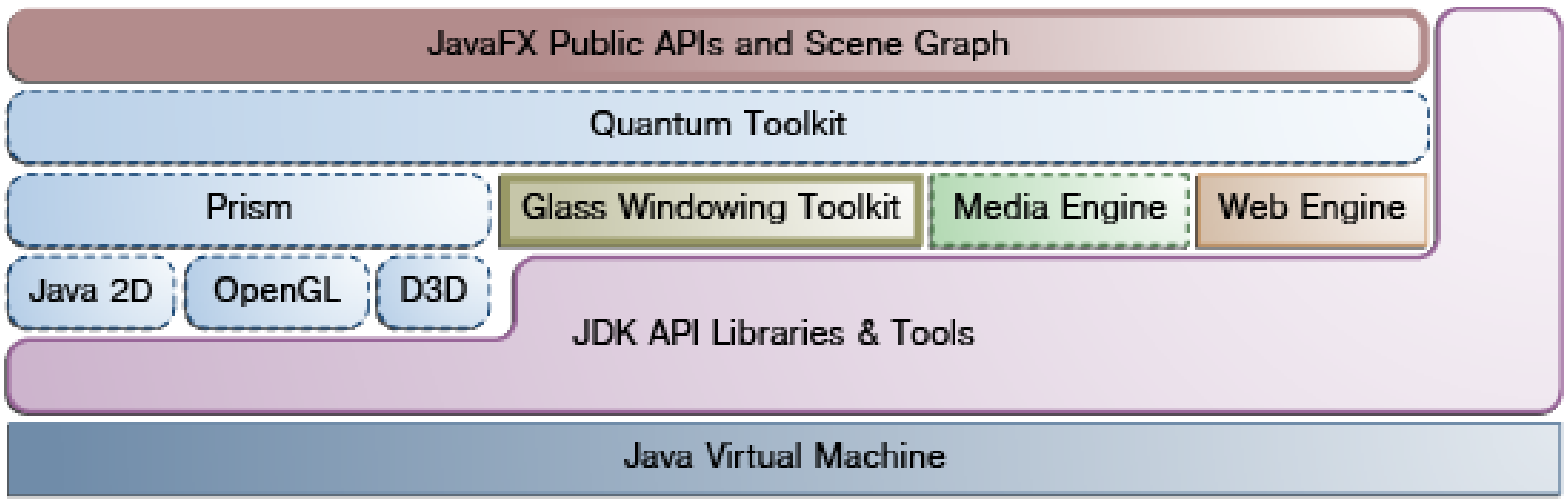
Model-View-Controller

- Design Pattern: Model-View-Controller (MVC)
- Entkopplung von
  - **Model**: Programmlogik und Datenspeicherung,
  - **View**: Darstellung der Daten und Realisierung der Benutzerinteraktion
  - **Controller**: Interaktion zwischen Model und View



JavaFX Grundlagen I

- Architektur:



JavaFX Grundlagen II

- `Application`: Hauptklasse einer JavaFX-Applikation
- `Stage`: Top-Level Container der Anzeige, kriegt eine `Scene` zugewiesen, die angezeigt wird
- `Scene`: wichtigstes Konzept für Erstellen von Anzeigen
  - Repräsentiert alle am Bildschirm sichtbaren Objekte
  - Hierarchische Baumstruktur
  - Jede `Node` ein visuelles Objekt
  - Hierarchische Struktur erleichtert Erstellung und Veränderung von Objekten (z.B. Animation)

JavaFX Grundlagen II

```
public class Main extends Application {
  public void start(Stage primaryStage)
      throws Exception{
    Parent root = new StackPane();
    Scene scene = new Scene(root, 300, 200);
    primaryStage.setTitle("Hello World");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

JavaFX Grundlagen III

- Threads:
  - **JavaFX application thread**.
    - Haupt-Thread für Grafikdarstellung
    - Alle sichtbaren Scene Graphs müssen hier aktiviert werden
    - Erstellung von Scene Graphs in anderem Thread möglich
      $\Rightarrow$ flüssige Darstellung
  - **Prism render thread**.
    - Ein oder mehr Threads für flüssiges Rendering von Szenen
  - **Media thread**.
    - Synchronisiert aktuelle Szenen mit JavaFX application thread.
- Rechenaufwendige Tasks in eigenen Thread auslagern!

```
Task task = new Task<Void>() {
    @Override public Void run() {
        // some work in own thread
        ...
        // update in JavaFX application thread
        Platform.runLater(() -> txtLabel.setText(...));
        return null;
    }
};
new Thread(task).start();
```

JavaFX Grundlagen IV

- Update der Anzeige: Automatisch bei jedem *Pulse*
  - Teil des Glass Windowing Toolkit
  - Maximal 60 fps
  - Wird automatisch gefeuert wenn Scene Graph sich verändert
  - Liest aktuelles CSS und Layout/Scene Graph und updated Anzeige entsprechend
  - Änderungen werden nur bei Pulse aktiv
    $\Rightarrow$ verhindert Performance-Einbrüche durch viele kleine Änderungen

View: Komponenten und Aktionen

- Standard-Komponenten:
  - Controls
  - Panes und Container
  - Transformationen
  - Visuelle Effekte

Controls

- Elemente im SceneGraph

Panes and Container

- Zum Layouten und Gruppieren von Controls
  - `Group`
  - `HBox`/`VBox`
  - `BorderPane`
  - `GridPane`
  - `...(tinyurl.com/JavaFXLayouts)`

Transformationen

- Verschieben
- Skalieren
- Rotieren
- . . . (`tinyurl.com/JavaFXTransf`)

Visuelle Effekte

- Schatten
- Reflektionen
- Lichter
- `tinyurl.com/JavaFXEffects`

Erstellung eines GUI

- Möglichkeiten zur Beschreibung des GUI:
  - Scene Graph programmatisch in Java oder XML-basiert mit FXML
  - WYSIWYG-Editor SceneBuilder generiert FXML
  - Styling mit CSS (ca. Version 2.1)
    `tinyurl.com/cssJavafx`

GUI-Beschreibung: Programmatisch

- Packages `javafx.scene.*`

```
GridPane root = new GridPane();
root.setAlignment(Pos.CENTER);

Label textLabel = new Label("Press the button!");
GridPane.setRowIndex(textLabel, 0);
GridPane.setColumnIndex(textLabel, 0);

Button btn = new Button("Say 'Hello World'");
btn.setOnAction(event -> {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Hello");
    alert.setHeaderText("A short message for you");
    alert.setContentText("Hello World!");

    alert.showAndWait();
});
root.add(textLabel, 0, 0);
root.add(btn, 0, 1);

Scene scene = new Scene(root);
primaryStage.setTitle("Hello World");
primaryStage.setScene(scene);
primaryStage.show();
```

GUI-Beschreibung: FXML

- GUI-Beschreibung in `.fxml`-Datei (hier: `sample.fxml`)
- GUI-Events in eigener Java Datei
- XML Laden in Main-Java-Datei

```
public class Main extends Application {

  @Override
  public void start(Stage primaryStage) throws Exception {
    Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
    Scene scene = new Scene(root);
    primaryStage.setTitle("Hello World");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

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

GUI-Beschreibung: FXML

- FXML:

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.*?>
<GridPane fx:controller="Controller"
      xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">
  <Label text="Press the button!"
         GridPane.columnIndex="0" GridPane.rowIndex="0"/>
  <Button text="Say 'Hello World"
         onAction="#handleSubmitButtonAction"
         GridPane.columnIndex="0" GridPane.rowIndex="1"/>
</GridPane>
```

- Controller:

```
public class Controller {
    @FXML
    void handleSubmitButtonAction(ActionEvent actionEvent) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Hello");
        alert.setHeaderText("A short message for you");
        alert.setContentText("Hello World!");

        alert.showAndWait();
    }
}
```

Skinning mit CSS

- Programmatisch:
  `scene.getStylesheets().add("simple.css");`
- FXML: Tag `<stylesheets>` innerhalb des Layouts

```
<GridPane fx:controller="sample.Controller"
          xmlns:fx="http://javafx.com/fxml">
    <stylesheets>
        <URL value="@sample.css"/>
    </stylesheets>
    ...
</GridPane>
```

Skinning mit CSS

- Klassen für Komponenten existieren meist,
  z.B. `.root`, `.button`, `.check-box`
- Eigene Klassen und IDs:
  - Programmatisch:
    ```
    btn.getStyleClass().add("input-buttons");
    greeting.setId("welcome-text")
    ```
  - FXML: Tag-Attribute `styleClass` und `id`
- Style kann auch programmatisch gesetzt werden:

```
btn.setStyle("-fx-background-color: slateblue; -fx-text-fill: white;");
```

- Referenz JavaFX CSS: `tinyurl.com/CssRefFX`

Skinning mit CSS: Beispiel

FXML-Datei:

```
<GridPane fx:controller="sample.Controller"
      xmlns:fx="http://javafx.com/fxml">
  <stylesheets>
    <URL value="@sample.css"/>
  </stylesheets>
  <Label id="welcome" fx:id="greetingText" text="Press the button!"
       GridPane.columnIndex="0" GridPane.rowIndex="0"/>
  <Button styleClass="input-buttons" text="Say 'Hello World"
        onAction="#handleSubmitButtonAction"
        GridPane.columnIndex="0" GridPane.rowIndex="1"/>
</GridPane>
```

CSS-Datei:

```
.GridPane {
    -fx-alignment: center;
    -fx-hgap: 10px;
    -fx-vgap: 10px;
}
.root {
    -fx-font-family: "Courier New";
    -fx-base: rgb(132, 145, 47);
    -fx-background: rgb(225, 228, 203);
}
#welcome {
    -fx-font-size: 16pt;
    -fx-padding: 0 3 5 10;
}
.input-buttons {
    -fx-padding: 5;
}
```

Controller (MVC)

- In FXML über Controller-Klasse und XML-Attribute (z.B. `onAction="#resize"`)
- Programmatisch: `resizeButton.setOnAction(event -> resize());`
- Wichtiges Konzept für automatische Updates/Observable-Pattern: `Properties`
  - Jede Komponente hat bestimmte Properties, z.B.: `widthProperty`, `translateXProperty`, `valueProperty`
  - Können `ChangeListener` erhalten oder gebunden werden

```
stage.widthProperty().addListener((obs, old, newVal) -> handleWindowResize());

speedSlider.valueProperty().addListener(new ChangeListener<Number>() {
  @Override
  public void changed(ObservableValue<? extends Number> obs, Number old, Number newVal)
    speed = MAX_SPEED / newVal.doubleValue();
  }
});
// Alternative to above ChangeListener:
speed = new DoubleConstant(MAX_SPEED).divide(speedSlider.valueProperty());
```

Weitere Resourcen

- Liste/Beschreibung aller UI-Komponenten:
  `tinyurl.com/FXControls`
- JavaDoc (für JavaFX 11):
  `openjfx.io/javadoc/11/`

Erweiterungen

- Seit JavaFX 11: Robot API für Testen/Simulation
  `tinyurl.com/JavaFxRobot`
- FormsFX: Framework zum Erstellen von Formularen in JavaFX
  `github.com/dlemmermann/FormsFX`
- BootstrapFX: Bootstrap-Style und Widgets für JavaFX
  `tinyurl.com/bootstrapfx`
- FXyz3D: 3D-Rendering mit JavaFX
  `github.com/FXyz/FXyz`
- ...