

# Workshop GUI mit FXML

## Einleitung

*Im folgenden Workshop wird eine Applikation erstellt. Diese Applikation stellt keine Ansprüche an ein in der Praxis taugliches Tool.*

*Es geht darum, dass Sie eine JavaFX Applikation von Grund auf erstellen und zur Darstellung FXML einsetzen können. Sie sind angehalten mitzudenken, die Schritte in Ihrer Umgebung auszuführen, auch wenn die Lösung jeweils gezeigt wird.*

## Ziele

- Sie erstellen ein neues Gradle Projekt von Grund auf.
- Sie probieren den JavaFX SceneBuilder aus.
- Sie erkennen, wie FXML Dateien mit den Klassen zusammenarbeiten.

## Voraussetzungen

- Vorlesung: GUI Foundations, GUI Toolbox

## Tooling

- Installiertes JDK 17+
- Gradle 7.3.3+
- JavaFX SceneBuilder

## SceneBuilder

SceneBuilder ist ein Werkzeug zur Bearbeitung von FXML-Dateien und ermöglicht somit die Erstellung und Bearbeitung eines SceneGraphs und die Verknüpfung mit einer zugehörigen Controller-Klasse.

SceneBuilder muss als eigenständiges Werkzeug heruntergeladen und installiert werden.



IDE's können den SceneBuilder einbinden (FXML-Code-Ansicht & SceneBuilder-Ansicht), bzw. liefern bereits eine Version von SceneBuilder mit (IntelliJ). Diese bietet jedoch meist nur einen eingeschränkten Funktionsumfang, weshalb es bei intensiver Arbeit trotzdem Sinn macht, die unabhängige Version zu installieren und aus der IDE aufzurufen (Dateipfad in Einstellungen konfigurieren und mit Kontextmenü **Open In SceneBuilder** öffnen)

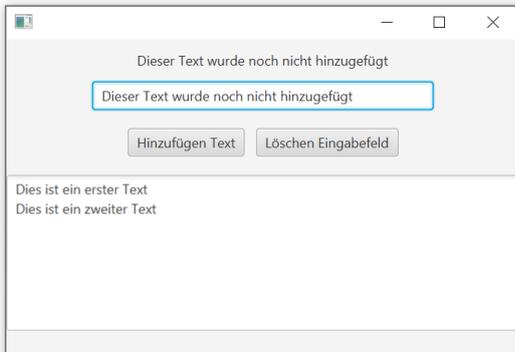
## Vorgehen

Schrittweise wird ein neues, einfaches Projekt angelegt. Dabei wird von einer Vorgabe (was soll die Applikation können, wie soll sie aussehen) ausgegangen.

Zusätzlich falls gewünscht: Das Resultat kann später im Praktikum weiterverwendet, mit einem Model versehen und gemäss MVC-Pattern verbunden werden.

# Umsetzung der Applikation

Erstellen einer App, die auf zwei Buttons reagiert, eine Texteingabe erlaubt und einen gebundenen Titel enthält. Die vorgeschlagene Lösung ist eine mögliche unter vielen. Es geht darum, einige Grundlagen zu zeigen.



## Was tut die App?

- "Hinzufügen Text" schreibt den Text aus dem Eingabefeld in die History.
- "Löschen Eingabefeld" leert das Eingabefeld.
- Der Titel über dem Eingabefeld ist gekoppelt mit dem Eingabefeld und zeigt den gleichen Inhalt.
- Die Eingaben werden nicht in einem Model gespeichert.

## Schritt 1: Erstellen des Projekts

Falls Sie bei einer Aktion nicht mehr wissen, wie das geht, schauen Sie in den Unterlagen nach...

1. Anlegen des Repositories «PROG2\_JavaFX\_WordCloud» auf GitHub, mit `.gitignore` für Java
2. Klonen des Repos in Ihr Working-Directory
3. Ausführen von `gradle init` (Terminal, bash, gitBash)

```
gradle init --type java-application \
--test-framework junit-jupiter \
--dsl groovy \
--project-name PROG2_JavaFX_WordCloud \
--package ch.zhaw.prog2.application
```

Eventuell werden, je nach verwendeter Gradle-Version, noch andere Werte abgefragt. Oder Sie verwenden `gradle init` und beantworten dann die Fragen.

4. Version ins lokale Repository schreiben und auf GitHub pushen
5. Öffnen Sie das Projekt in Ihrer bevorzugten IDE
6. Projekt für JavaFX fit machen durch das Ergänzen von `build.gradle` (siehe auch die Anleitung für Gradle auf der OpenJFX Website)

```
plugins {
    // ...
    // Adding JavaFX support and dependencies
    id 'org.openjfx.javafxplugin' version '0.0.12' ①
}

// Configuration of the JavaFX plugin
javafx { ②
    version = '17' ③
    modules = [ 'javafx.controls', 'javafx.fxml' ] ④
}
```

- ① Hinzufügen des Gradle-Plugins für JavaFX
- ② Die Konfiguration des Plugins erfolgt im entsprechenden Block
- ③ Zum einen können Sie die spezifische JavaFX-Version angeben, die verwendet werden soll. Diese sollte natürlich zur verwendeten Java-Version kompatibel sein.
- ④ Zum Anderen können sie spezifizieren, welche JavaFX-Module eingebunden werden sollen.

Damit sollte das Projekt bereit sein. Eventuell ist noch ein IDE-Refresh für Gradle notwendig, damit die Anpassungen am `build.gradle` wirksam werden.

## Schritt 2: Vorbereiten der JavaFX Applikation



Beim Start einer JavaFX-Applikation wird jeweils ein «Application» Objekt erstellt. In diesem Objekt läuft der Main-Loop, bis es wieder geschlossen wird. Die Klasse dieses Objekts erbt von `javafx.application.Application` und muss instantiiert werden über den Aufruf `Application.launch`, welcher dann implizit die `start()` Methode aufruft.

Aber, eines nach dem anderen:

- Den Aufruf von `Application.launch` weisen wir der durch die Gradle Initialisierung bereits vorhandenen App-Klasse zu, welche die Main-Methode enthält. Dafür müssen wir sie anpassen.
- Als Application Klasse erstellen wir eine neue Klasse, Name = `MainWindow`
  - nur mal so weit:

```
public class MainWindow extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
  
    }  
}
```

- Und als Anpassung in der Klasse `App` (falls nicht vorhanden bitte erstellen)

```
public class App {  
    public static void main(String[] args) {  
        Application.launch(MainWindow.class, args);  
    }  
}
```

### Warum eine eigene Klasse App für den Start?

Eine separate Main-Klasse ist notwendig, um die Java-FX-Anwendung aus der IDE zu starten. Ansonsten wird ein Runtime-Fehler ausgegeben: "Fehler: Zum Ausführen dieser Anwendung benötigte JavaFX-Runtime-Komponenten fehlen". Wegen der Java-Modularisierung müssen die JavaFX-Module geladen werden, bevor eine Instanz von Typ `javafx.application.Application` erstellt werden kann. Dies kann auf 3 Arten gemacht werden:



- Wie in diesem Beispiel: `main()` in separate Klasse und die Anwendung mit der statischen `launch()` Methode. Diese initialisiert das JavaFX-Modul, bevor die Instanz der übergebenen Klasse erzeugt wird.
  - Start der Anwendung mit `gradle run`. Die JavaFX-Module werden vom Gradle `javafxplugin` vor dem Start geladen.
  - Konfiguration von JavaFX in der IDE und im run command (VM Options) des Projektes wie hier erläutert: <https://openjfx.io/openjfx-docs/#IDE-Intellij>
- Nun soll die start-Methode tatsächlich ein Fenster öffnen.  
Dazu erstellen wir als Vorbereitung eine Methode in der Klasse `MainWindow`, welche diesen Vorgang später ausführen wird.
- Neue Methode erstellen, Signatur = `private void openMainWindow(Stage stage)`

#### Aktueller Stand:

```
public class MainWindow extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        openMainWindow(primaryStage);  
    }  
  
    private void openMainWindow(Stage stage) {  
  
    }  
}
```

## Schritt 3: SceneBuilder ausprobieren

Damit steht dem ersten Fenster nichts mehr im Wege. Ausser, dass wir noch keinen Inhalt definiert haben. Dies wird nun über eine FXML-Datei und SceneBuilder gemacht.

Als Basis braucht SceneBuilder eine FXML-Datei, die Sie am einfachsten aus der IDE heraus erstellen. Der passende Ort ist das `resources`-Verzeichnis, als Name bietet sich `MainWindow.fxml` an.



Wählen Sie beim Erstellen als "Root Element" ein `javafx.scene.layout.AnchorPane`.

Nachdem die Datei erstellt ist, öffnen Sie diese mit SceneBuilder (über das Kontextmenü der IDE). Sie sehen eine Grundeinstellung. Probieren Sie die verschiedenen Einstellungen und Menüs aus.

Verwenden Sie die Suche um Elemente aus der Bibliothek zu zeigen.

Linke Spalte: Zeigt die platzierten Elemente und die Auswahl für weitere

Darstellungsbereich für die Container und die Controls. Aktuell: Leerer Container vom Typ AnchorPane (Grösse 0).

Rechte Spalte: Zeigt die Eigenschaften des gewählten Elements.

Verwenden Sie die Suche um Eigenschaften einzuschränken.

Umstellen der Anzeige. Z.B. auf eine Liste aller Elemente.

Umstellen der Anzeige. Z.B. auf alphabetische Liste der Einstellungen.

Alles, was das Element als Ganzes betrifft.

Der Scene-Graph dieser Darstellung wird hier angezeigt.

Darstellung des Elements.

Verbindung zum Controller und eine Liste aller im Controller verwendbaren Elemente.

Einstellungen des Elements zur Zusammenarbeit mit dem Controller.

**Zum Ausprobieren:**

- Vergrössern Sie den Platz für das Anchor-Pane auf 640x480 Pixel.

Ziehen der Preferred Size mit der Maus

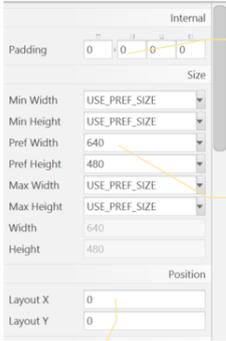
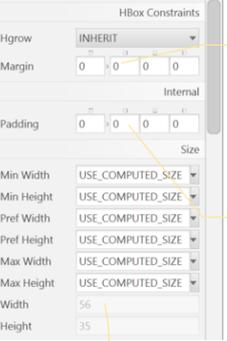
Einstellen der Preferred Width und Height in den Layout-Eigenschaften

- Platzieren Sie einen HBox Container im Anchor-Pane.

Per Drag&Drop ein Element im Scene Graph platzieren

Alternativ direkt per Drag&Drop in den Zeichenbereich

- Platzieren Sie ein Label-Element in den HBox Container.
- Schalten Sie nun zwischen den einzelnen Elementen um, indem Sie sie im Scene Graph anwählen.  
Beachten Sie dazu rechts die Layout-Eigenschaften:

<i>Layout Eigenschaften von AnchorPane</i>	<i>Layout Eigenschaften von HBox</i>	<i>Layout Eigenschaft von Label</i>
 <p>Internal wirkt sich auf die Unter-elemente aus. Padding stellt den Abstand der Unter-elemente vom Rand ein.</p> <p>Grösse der Elements. USE PREF SIZE übernimmt die Werte aus Pref. Anchor Pane ist hier Root Node. Also wird ohne Einstellung die Grösse der Scene übernommen.</p> <p>Die Position legt fest, wo auf dem Ober-element das Element angezeigt wird. Beim Root Element spielt das eine untergeordnete Rolle.</p>	 <p>Stellt ein, an welcher Seite und mit welchem Abstand die HBox verankert wird. Gegenüberliegende Abstände mit 0 nutzen die ganze Breite fix aus.</p> <p>Stellt ein, wie sich Unter-elemente verhalten. Padding bestimmt den Abstand vom Rand, Spacing den Abstand zwischen den Unter-elementen.</p> <p>Grösse der HBox. USE COMPUTED SIZE stellt den Platz gemäss den Verhältnissen der Ober-elemente ein oder braucht so viel Platz, wie die Unter-elemente belegen.</p>	 <p>Margin: Abstände zum Rand und anderen Elementen. Siehe Padding des Ober-elementes (hier HBox). Hier nur abweichend einstellen.</p> <p>Padding: Label ist ein Control und kann kein Unter-elemente haben. Der Text wird aber gleich behandelt. Also: Abstand des Textes vom Rand.</p> <p>Zeigt die aktuelle Grösse des Labels an. USE COMPUTED SIZE berücksichtigt die Textgrösse und alle Einstellungen von Margin und Padding.</p>

## Schritt 4: Layout erstellen

Erstellen wir nun das abgemachte Layout. Dazu analysieren wir die Vorgabe:

- Es gibt 4 Zeilen
  - Zuerst ein Label,
  - dann ein Texteingabefeld,
  - auf der nächsten Zeile zwei Buttons nebeneinander
  - und ab ca. der vertikalen Hälfte ist der untere Teil von einem Textanzeigefeld ausgefüllt.
- Über das Verhalten beim Vergrössern und Verkleinern des Fensters wissen wir nichts.



### Vorschlag

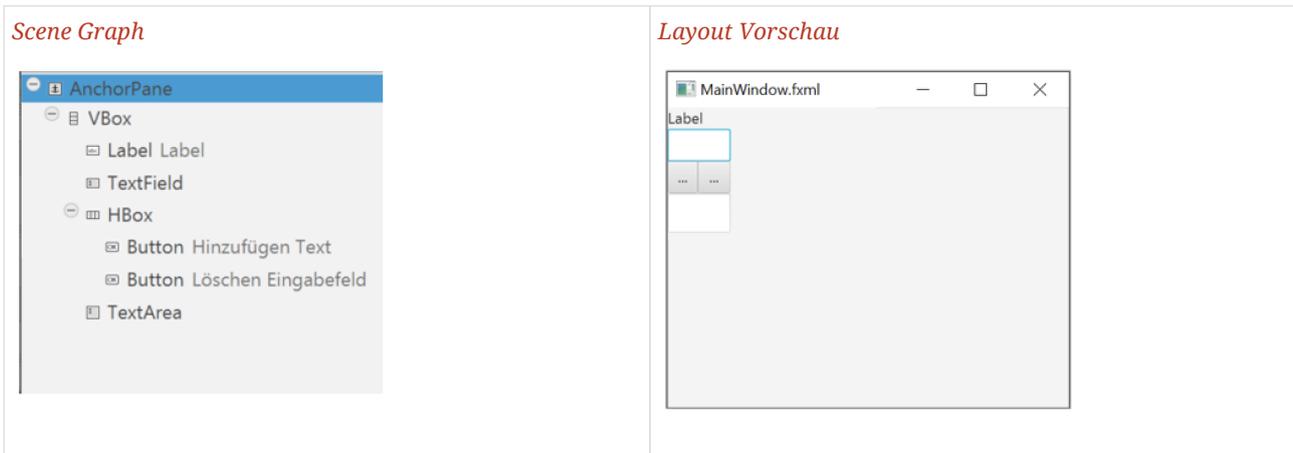
Die vier Zeilen in einer VBox abbilden. Diese VBox enthält zuerst ein Label, dann das Textfeld (TextField). Als Nächstes eine HBox für die beiden Buttons und als letztes Element die TextArea.

Nach den Übungen von vorhin sollte das kein Problem darstellen. Entfernen Sie die Testelemente aus der AnchorPane und fügen Sie die 6 Elemente ein. Beschriften Sie auch gleich die Buttons entsprechend mit "Hinzufügen Text" und "Löschen Eingabefeld" (Einstellungen in den Properties der Buttons). Es braucht etwas Übung mit der Maus.



- Vergessen Sie nicht, zwischendurch zu speichern (**File** > **Save**).
- Ausserdem können Sie mit **Preview** > **Show Preview in Window** das Resultat jederzeit kontrollieren.

**Das Layout sieht noch etwas dürftig aus:**



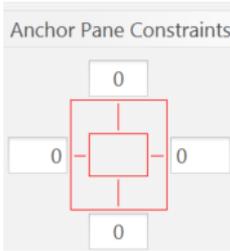
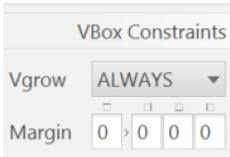
Das Anchor Pane ist zwar wie eingestellt 640x480 Pixel gross, aber die Elemente kleben alle oben links, die Texte auf den Buttons sind nicht lesbar.

Versuchen Sie nun mit den Einstellungen des Layouts das vorgegebene Aussehen zu erreichen.  
**Experimentieren Sie!**



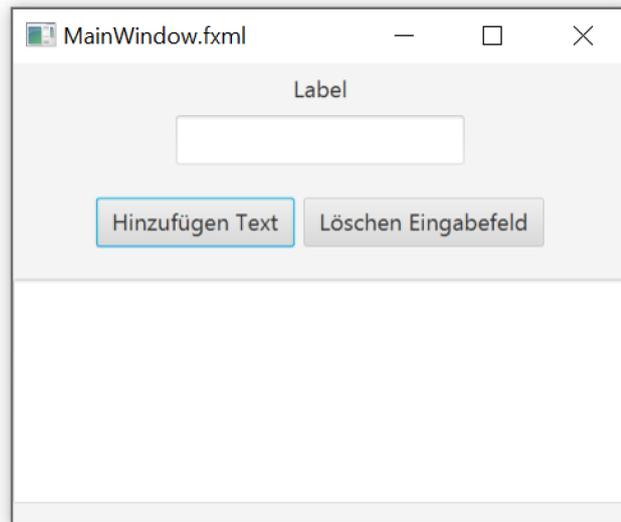
Es gibt nicht nur einen Satz von möglichen Einstellungen, der zum Ziel führt. Es ist jeweils die Kombination von Einstellungen, wobei Sie immer überlegen müssen, auf welche Elemente sich eine Einstellung auswirkt (z.B. auf das Element selbst oder auf alle Unterelemente).

Die folgenden Einstellungen führen ebenfalls zum gewünschten Aussehen:

Einstellung	Bild	Auswirkung
<Alignment> der VBox selbst auf CENTER		Alle Elemente in der VBox werden horizontal zentriert
VBox innerhalb der AnchorPane		VBox an alle 4 Seiten anheften, damit sie den vollständigen Platz in der AnchorPane ausfüllt.
<Alignment> der HBox auf CENTER		Die HBox wurde zentriert in der VBox, nun noch die Elemente in der HBox zentrieren
TextArea <Vgrow> auf ALWAYS		Platz bis zum Ende der VBox füllen
TextField <Max Width> auf 300		Eingabefeldgrösse beschränken
VBox <Top Padding> auf 10		Label etwas vom oberen Rand wegnehmen

Einstellung	Bild	Auswirkung
VBox <Spacing> auf 10		Abstand der Elemente in der VBox
HBox <Spacing> auf 10		Abstand der Elemente in der HBox (Buttons)
AnchorPane <Bottom Padding> auf 20		Unterer Rand (könnte auch weggelassen werden, dann geht die TextArea bis zum unteren Rand).

Nun sieht das Resultat schon passabel aus:



So, das war der anspruchsvolle Teil. Speichern Sie die FXML Datei, dann fahren wir weiter mit dem technischen Teil...

## Schritt 5: Verbindung herstellen zu unseren Klassen

Die Applikation soll, wie in der Beschreibung am Anfang erwähnt, etwas tun. Dafür wird Code geschrieben, in Klassen, wie gewohnt. Wenn wir mit FXML arbeiten, nimmt uns JavaFX allerdings einen gewichtigen Teil ab: **Wir müssen uns nicht um das Event-Handling kümmern!**

Trotzdem sind einige Überlegungen (Fragen) notwendig:

- Welche Controls sollen im Javacode verwendet werden?
- Welche Controls führen eine Aktion aus?
- Wie heisst der Controller, der für dieses Fenster zuständig ist?

### Es gilt:



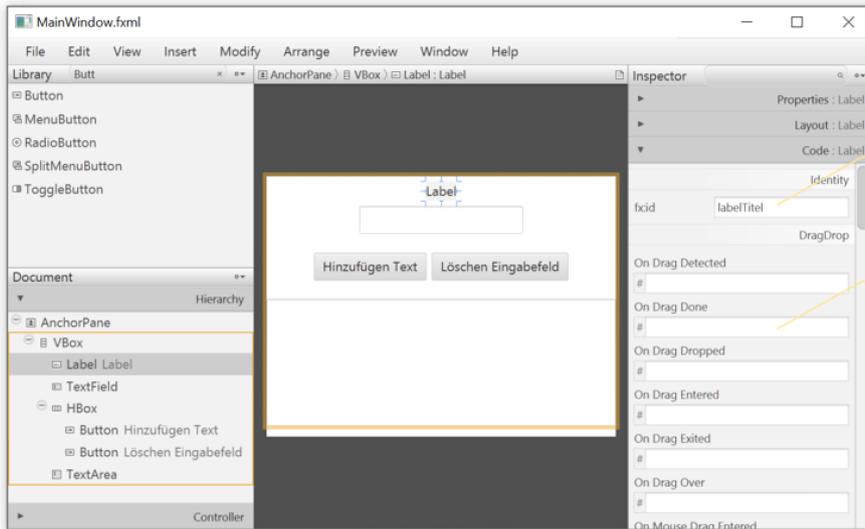
- Jedes Element, auf das wir im Code zugreifen wollen, muss eine eindeutige id bekommen.
- Von jeder Methode, die wir implementieren, um Aktionen zu behandeln, muss beim entsprechenden Event der Methodename hinterlegt werden, d.h. sie wird als Event-Handler-Methode verknüpft.



**Antworten:**

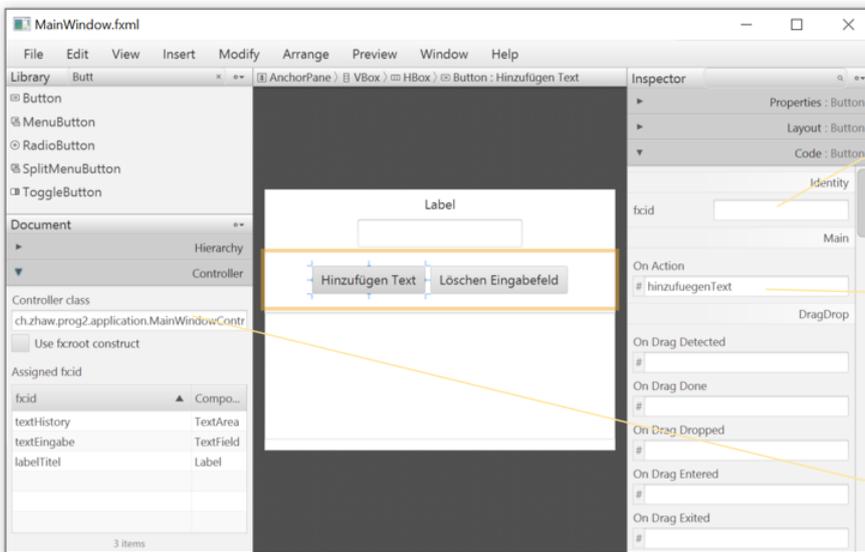
- Texte werden im Textfeld eingegeben: Textfeld benennen mit `textEingabe`
- Texte werden im Label Titel angezeigt: Label benennen mit `labelTitel`
- Texte werden in die TextArea kopiert: TextArea benennen mit `textHistory`
- Die Buttons lösen Aktionen aus: Methodennamen `hinzufuegenText` und `leerenTextEingabe` eintragen.
- Der Klassenname des Controller ist frei zu vergeben: `MainWindowController`

Im Abschnitt Code der Einstellungen finden Sie für jedes Element die Einstellmöglichkeiten.



Vergeben der fx:id für alle Elemente, die im Code verwendet werden sollen.

Hinterlegen von Methodennamen zu den passenden Ereignissen.



Falls nur eine Methode ausgeführt werden soll, ist die fx:id für das Element nicht notwendig.

Das Ereignis «On Action» wird bei einem Mausklick wie auch bei der Taste «Enter» auf dem Button ausgeführt.

Hier tragen Sie den Controller ein, der mit dieser FXML Datei geladen werden soll

Damit ist das Layout fertig vorbereitet (bitte speichern) und wir können mit dem Controller beginnen.



Damit wir nicht alles von Hand machen müssen, bietet SceneBuilder ein Skeleton für den Controller an. Sie finden das im Menü **View > Show sample controller skeleton**. Probieren Sie es aus und kopieren Sie den Inhalt in die Zwischenablage.

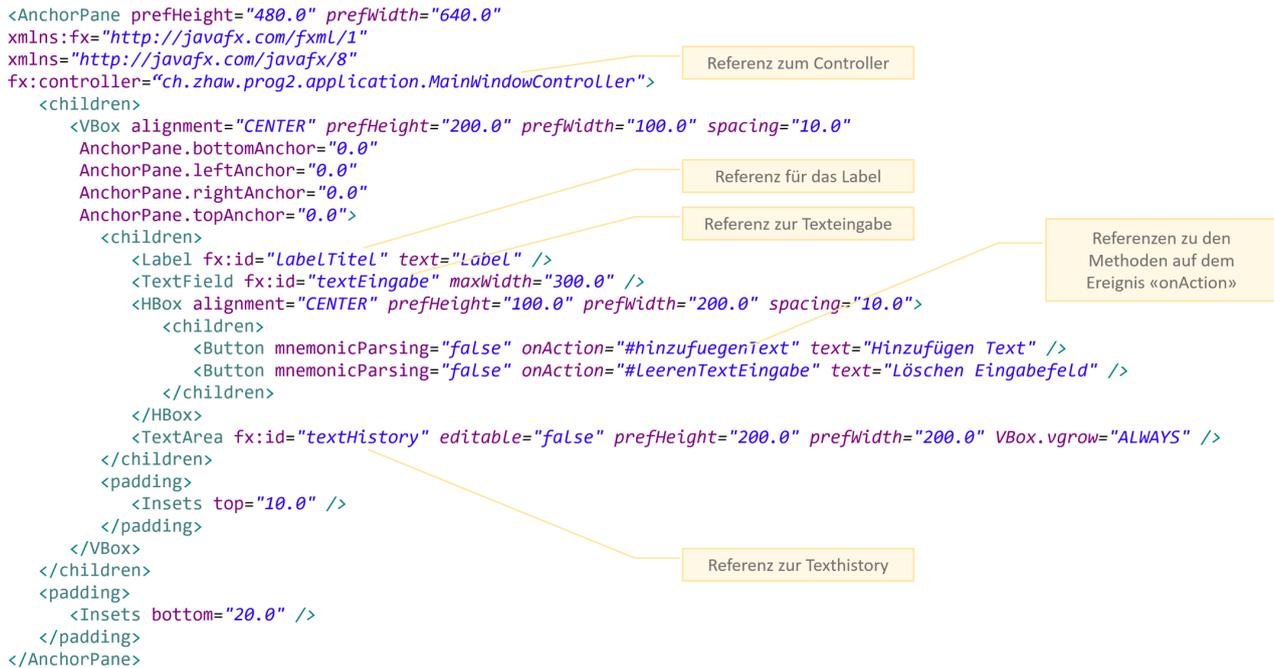
## Schritt 6: Erweitern der Klassen in der IDE

Wechseln Sie wieder in Ihre IDE und erstellen Sie dort eine Klasse mit dem Namen `MainWindowController`.

Sobald Sie die Klasse offen haben, können Sie die Zwischenablage reinkopieren.

Den SceneBuilder können Sie schliessen, die FXML Datei ist erstellt. Schauen Sie sich das Resultat an, indem Sie die Datei in Ihrer IDE als Textdatei öffnen:

Sie sehen eine einfache XML Datei mit den Angaben zur Darstellung und den Verbindungsinformationen zu den Klassen im Code.



Um diese Darstellung (das Öffnen des Fensters) ausprobieren zu können (ohne jede Funktionalität) bauen wir den Aufruf der FXML Datei sofort ein. Öffnen Sie dafür die vorbereitete Klasse `MainWindow` in der IDE.

Die Methode `openMainWindow` ist noch leer!

Wir benötigen einen Vorgang, welcher die Informationen aus der XML Datei lädt und die Verbindung zum Controller (den wir bereits vorbereitet haben) herstellt. Dafür bietet JavaFX die Klasse `javafx.fxml.FXMLLoader` an.

Je nachdem, wie Sie die Struktur des Projekts aufgebaut haben, finden sich die Dateien nicht gegenseitig. Der `FXMLLoader` muss die FXML Datei finden. Beim Laden muss er dann die Controller Klasse finden (hier `MainWindowController`), deren Referenz in der FXML Datei steht.

#### Finden der FXML Datei:

- Die FXML Datei befindet sich im Ordner `src/main/resources` und dort auf dem gleichen Package-Level wie die Applikation, z.B. `ch.zhaw.prog2.application` (Best Practice)
  - Greifen Sie direkt auf die Datei zu mit `FXMLLoader(getClass().getResource("MainWindow.fxml"))`
- Die FXML Datei befindet sich im Ordner "src/main/resources" und dort auf einem anderen Package-Level, z.B. "application"
  - Greifen Sie auf die Datei zu mit `FXMLLoader(getClass().getClassLoader().getResource("application/MainWindow.fxml"))`



#### Finden des Controllers aus der FXML Datei:

- Geben Sie immer den vollständigen Namen der Klasse an (inklusive Package). Aktuell z.B. `ch.zhaw.prog2.application.MainWindowController`.

Ergänzen Sie die `openMainWindow(Stage stage)` Methode um die 3 Schritte:

1. Loader für die FXML Datei erstellen

```
FXMLLoader loader = new FXMLLoader(... gemäss Hinweis...);
```

2. SceneGraph aus FXML Datei laden

```
Pane rootNode = loader.load();
```

3. Scene initialisieren, setzen und die Stage anzeigen

```
Scene scene = new Scene(rootNode);  
stage.setScene(scene);  
stage.show();
```



Falls alles korrekt eingegeben wurde, sollten Sie jetzt die Applikation mit `gradle.run` ausprobieren können.

Die Umsetzung der Methoden ist nun reine Formsache. Zur Erinnerung, was die Applikation tun soll:

- «Hinzufügen Text» fügt den Text aus dem Eingabefeld der bestehenden History an.
- «Löschen Eingabefeld» leert das Eingabefeld.
- Der Titel ist «gekoppelt» mit dem Eingabefeld und zeigt den gleichen Inhalt.
- Die Eingaben werden nicht in einem Model gespeichert.

## Codieren

Als Erstes verbinden wir den Titel mit dem Eingabefeld. Um diese Verbindung setzen zu können, brauchen wir eine Methode, die nach dem Instanzieren des Controllers (passiert beim `loader.load()` Befehl) im Controller ausgeführt wird.

Wir nennen diese Methode `connectProperties()`.

Wie Sie nun richtig vermuten, gehört diese Methode in den Controller!

Es gibt mehrere Möglichkeiten, Verbindungen zwischen Properties herzustellen:

- Über das Verbinden von Properties mit `bind`.  
Im aktuellen Fall binden wir das Text-Property von `textEingabe` an dasjenige von `labelTitel`:

```
labelTitel.textProperty().bind(textEingabe.textProperty());
```

- Über das Hinzufügen eines Listeners, der auf die Veränderung des Textes reagiert:

```
textEingabe.textProperty().addListener(new ChangeListener<String>()
{
    @Override
    public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue)
    {
        labelTitel.setText(newValue);
    }
});
```



Erstellen Sie die `connectProperties()` Methode.

Ergänzen Sie `MainWindow.openMainWindow` um den Schritt 2a, bei dem eine Instanz des `MainWindowController` geholt und auf dieser Instanz die Methode `connectProperties()` ausgeführt wird. Eine Instanz des Controllers erhalten Sie über die Methode `getController()` des `FXMLLoaders`.



Sollte auch das geklappt haben, können Sie ausprobieren, ob der Eingabetext jetzt mit dem Titel verbunden ist.

Die Methode `openMainWindwo(Stage stage)` sollte nun etwa so aussehen:

```
private void openMainWindow(Stage stage) {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("MainWindow.fxml"));
        Pane rootNode = loader.load();
        MainWindowController mainWindowController = loader.getController();
        mainWindowController.connectProperties();

        Scene scene = new Scene(rootNode);
        stage.setScene(scene);
        stage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Es folgt nun die Umsetzung der Methoden `hinzufuegenText` und `leerenTextEingabe`, die bereits im `MainWindowController` vorbereitet sind.

Die Methode `leerenTextEingabe` stellt kein Problem dar, die Klasse `TextField` bietet eine Methode `clear()` an.

Das Hinzufügen des Textes zu einem bestehenden Inhalt ist ebenfalls schnell geschrieben. Verwenden Sie dazu die Methoden `setText()` und `getText()` die auf einer `TextArea` und auf einem `TextField` vorhanden sind.



Die Applikation ist damit fertig. Sie macht noch nicht viel und speichert auch die Eingaben nicht in einem Model. Im Praktikum werden wir diese Aufgabe noch einmal aufgreifen und mit einem Model versehen.

Die Klasse `MainWindowController` ist denkbar einfach, weil wir das `EventHandling` nicht schreiben müssen:

```
package ch.zhaw.prog2.application;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

public class MainWindowController {
    @FXML
    private TextArea textHistory;
    @FXML
    private TextField textEingabe;
    @FXML
    private Label labelTitel;

    public void connectProperties() {
        labelTitel.textProperty().bind(textEingabe.textProperty());
    }

    @FXML
    private void hinzufuegenText(ActionEvent event) {
        textHistory.setText(textHistory.getText() + textEingabe.getText() + System.LineSeparator());
    }

    @FXML
    private void leerenTextEingabe(ActionEvent event) {
        textEingabe.clear();
    }
}
```