TEKO
SCHWEIZERISCHE FACHSCHULE

2D-Graphik & Frameworks

Teil 2: JavaFX & Frameworks

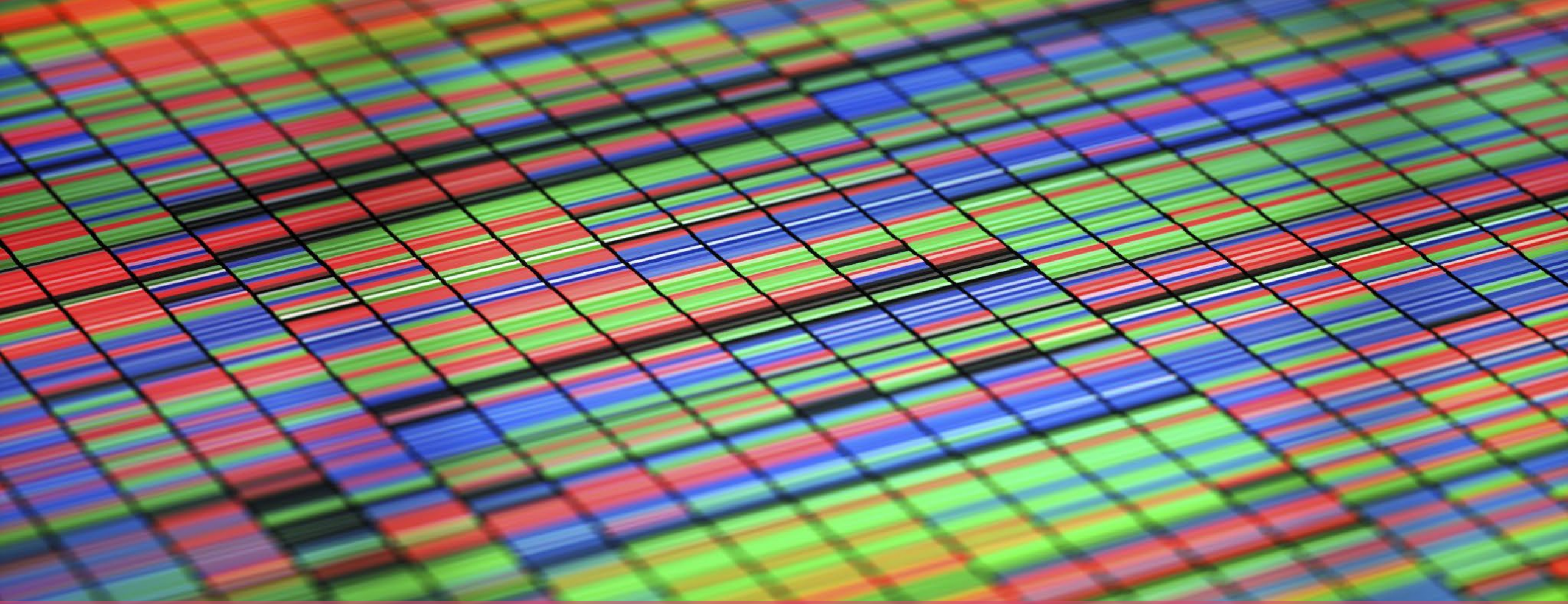
Björn Näf
Dozent

bjoern.naef@edu.teko.ch

Agenda

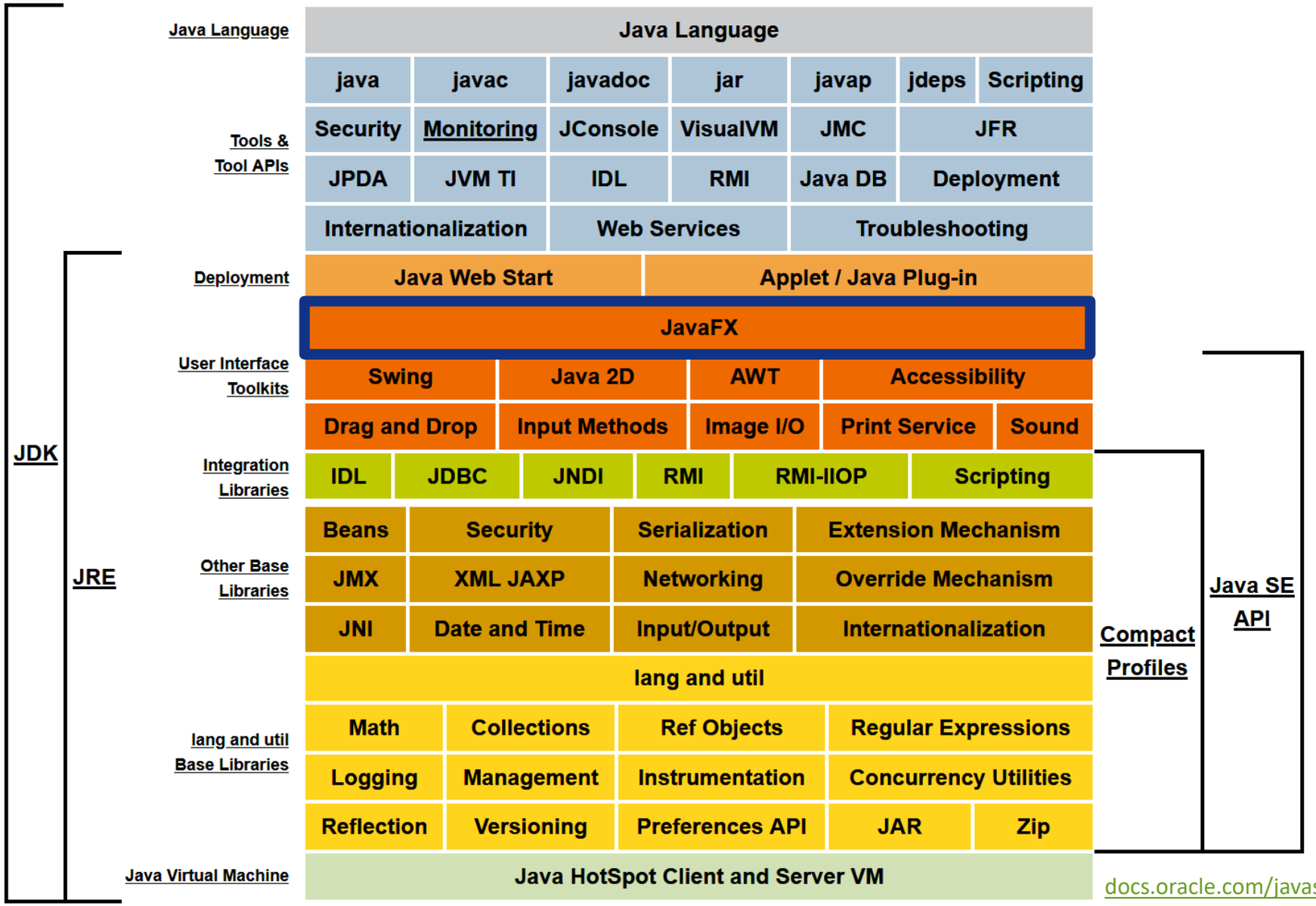
Teil 2

- **JavaFX**
  - Real World Beispiele
  - Erste Schritte
  - Layout
  - Dialoge
  - Vergleich zu AWT / Swing
- **Weitere Frameworks im Java-Ökosystem**
  - für verschiedene Anwendungszwecke



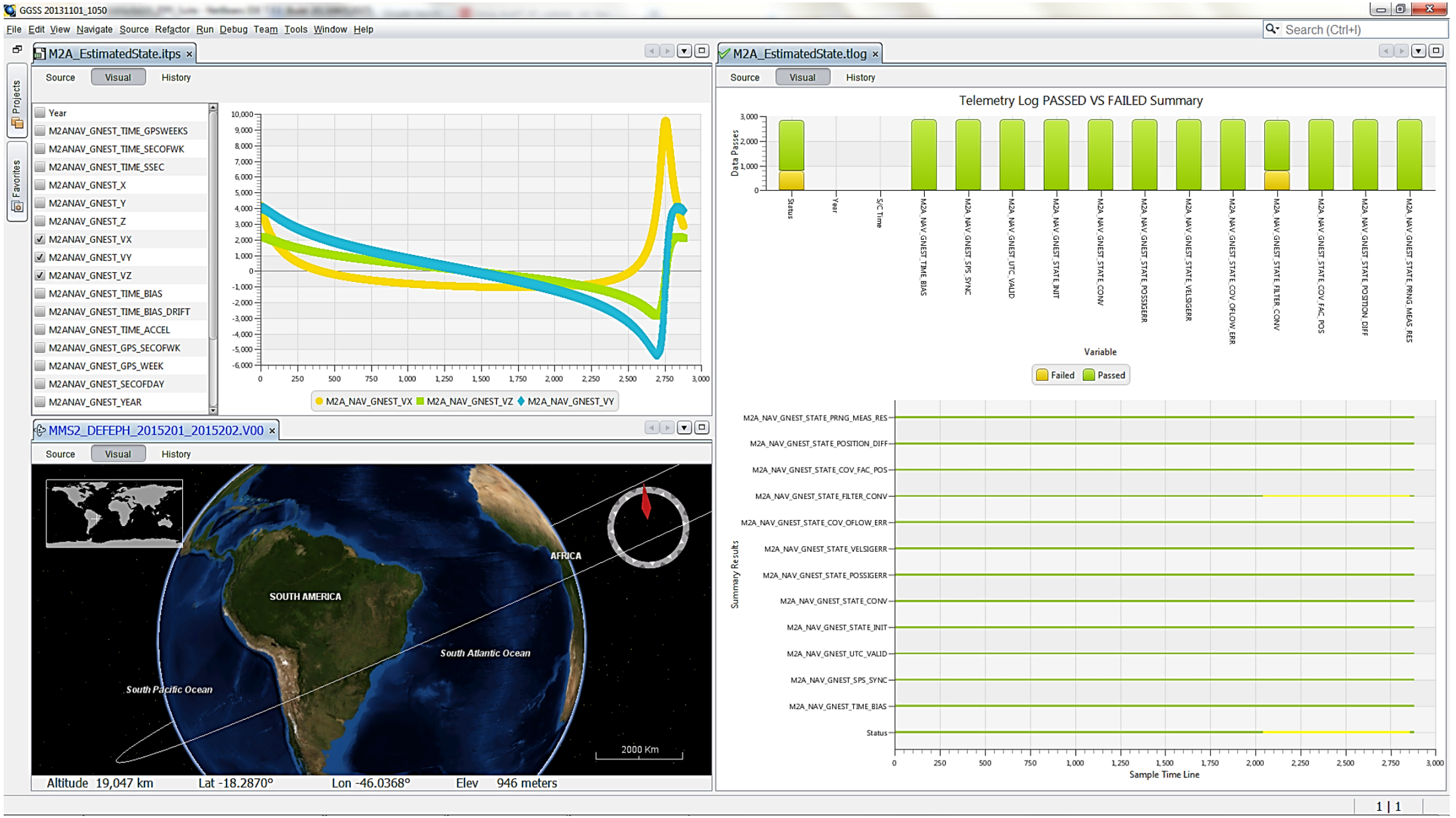
JavaFX – Modernde Desktopanwendungen gestalten

JavaFX: Einordnung im Java / JDK Framework



docs.oracle.com/javase/8/docs/index.html

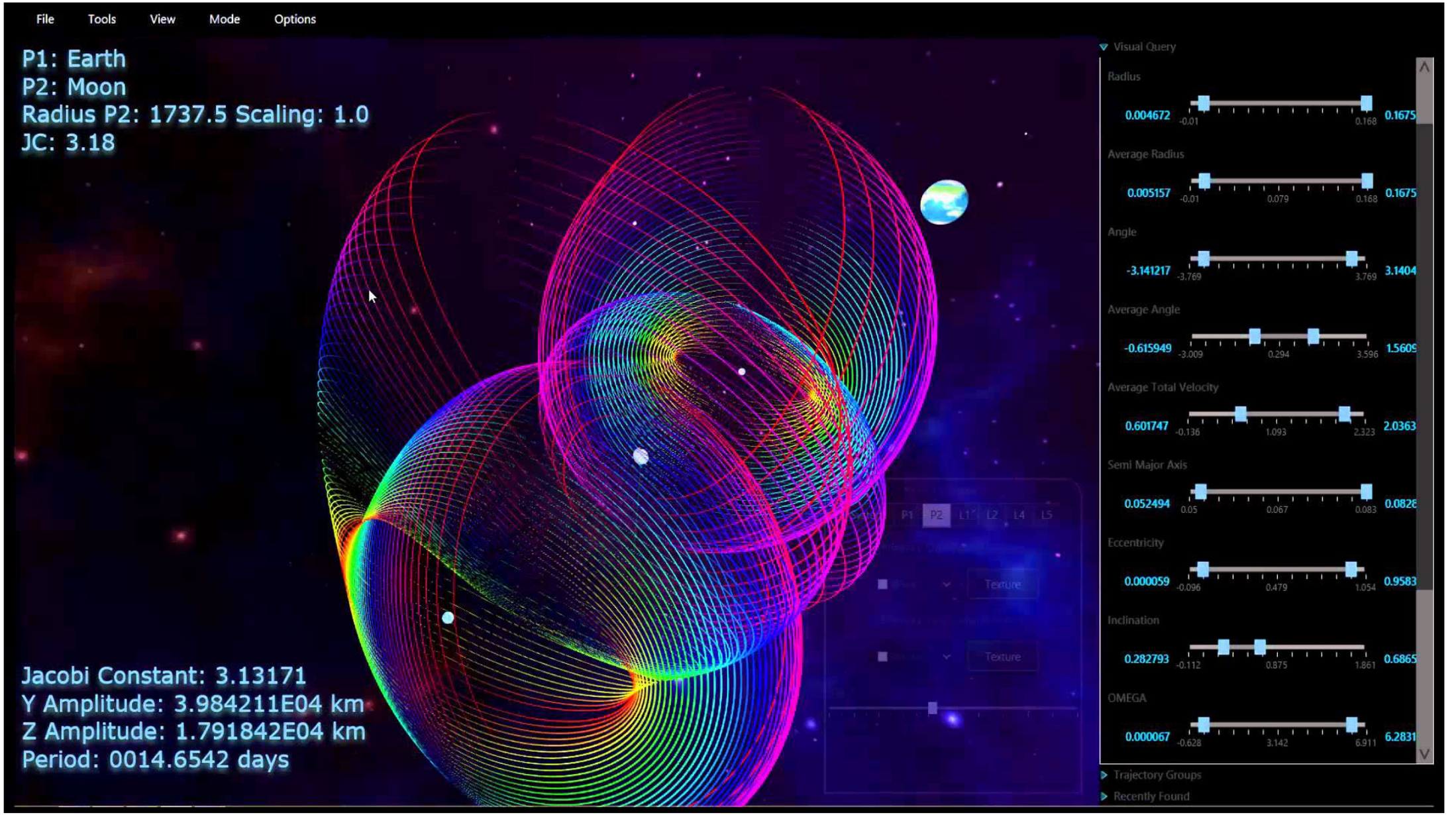
JavaFX Real World Beispiele



https://devm.io/java/20-javafx-real-world-applications-123653

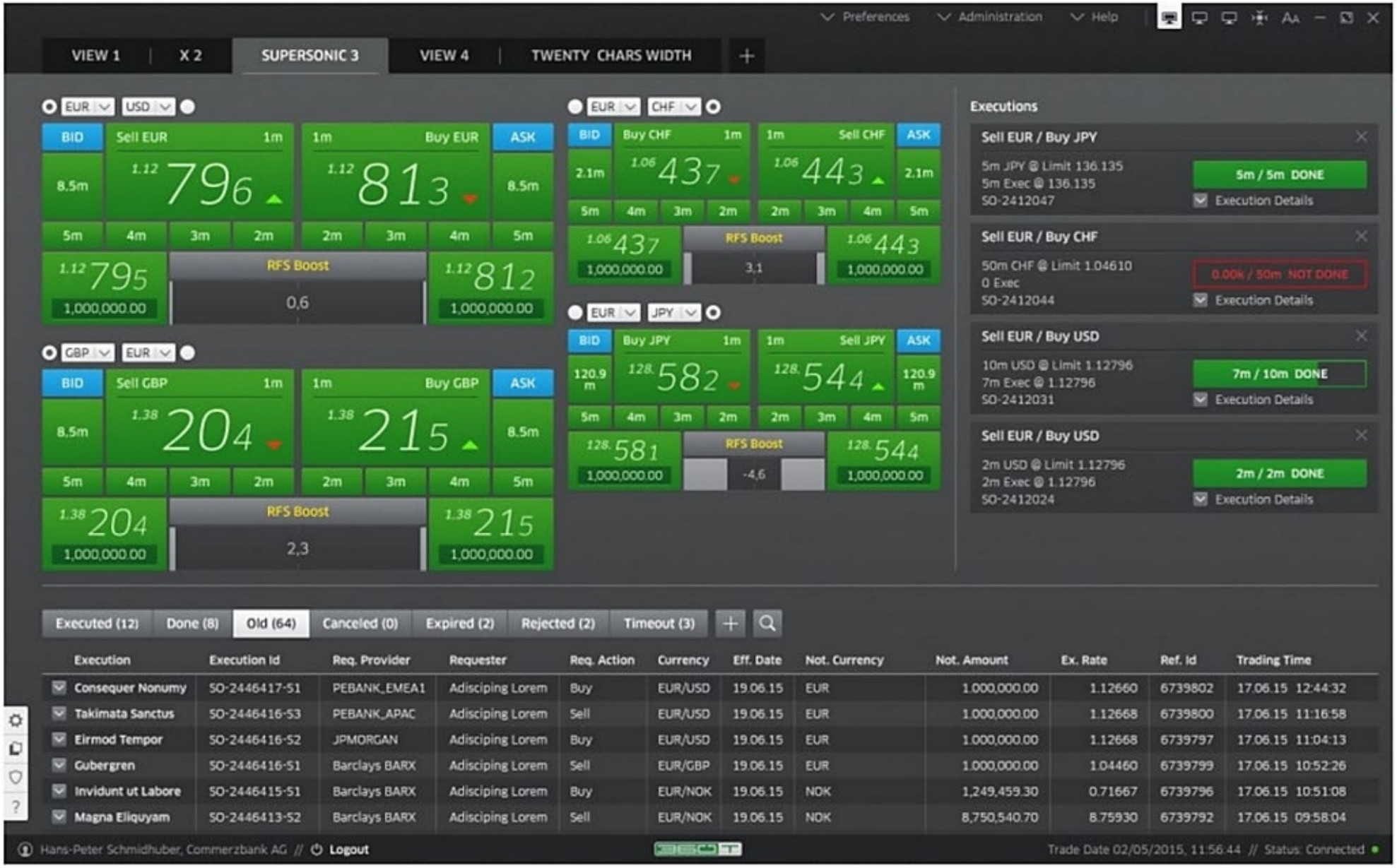
JavaFX Real World Beispiele

TEKO
SCHWEIZERISCHE FACHSCHULE



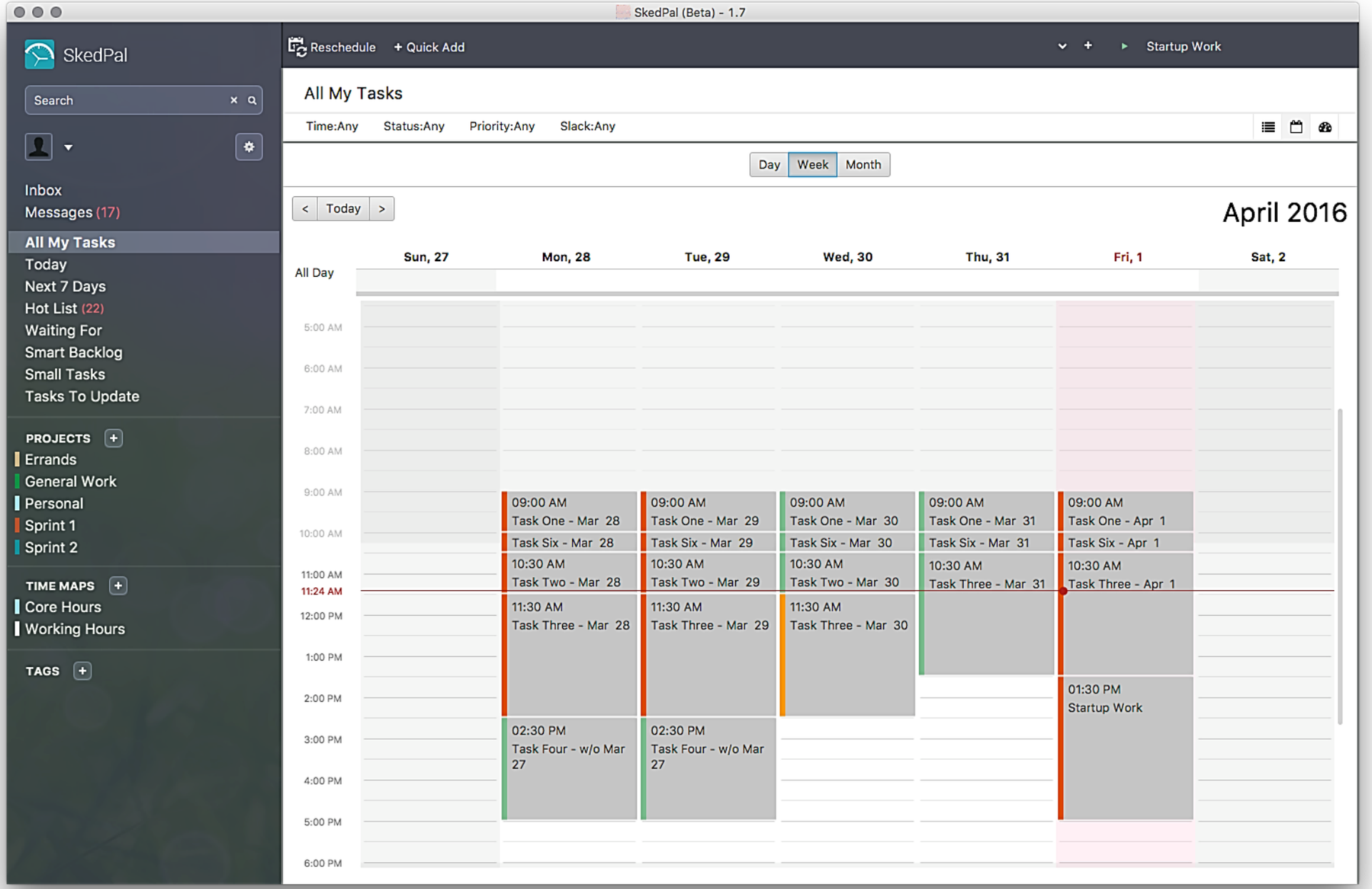
https://devm.io/java/20-javafx-real-world-applications-123653

JavaFX Real World Beispiele



https://devm.io/java/20-javafx-real-world-applications-123653

JavaFX Real World Beispiele



https://demos.jpro.one/calendar.html

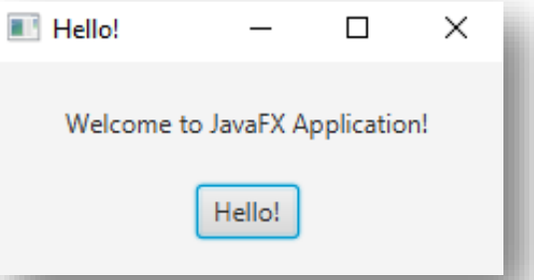
JavaFX: Getting Started…

Teil 1: JavaFX installieren, neues Projekt erstellen

- Wie zuvor erwähnt ist JavaFX grundsätzlich Teil des JDK Framework
  - Seit JDK 9 wird Java aber modularisiert ausgeliefert, d.h. es gibt nicht mehr „ein JDK für alles“
  - Seit JDK 11 ist JavaFX nicht mehr integriert, sondern muss u.U. separat bezogen bzw. installiert werden
  - JavaFX ist seither Open Source und modularisiert, muss u.U. in Teilen bezogen bzw. installiert werden

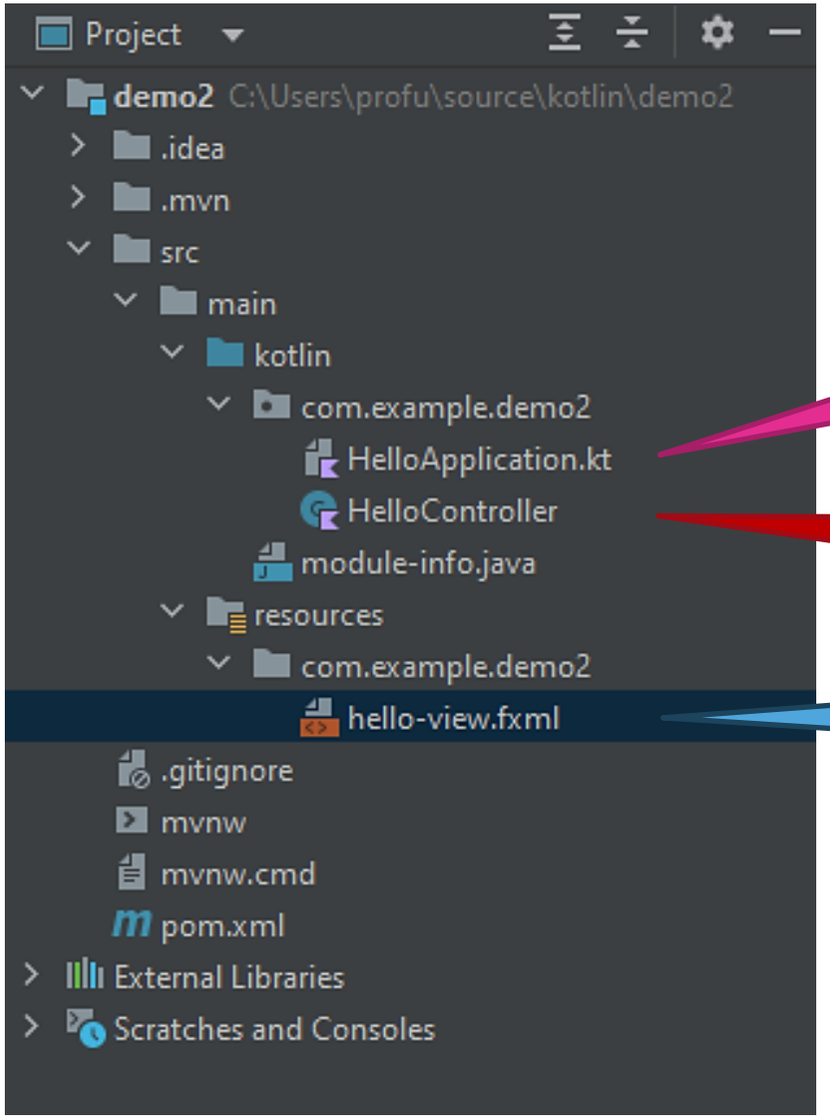
- Download: https://openjfx.io/ (Installationsverzeichnis notieren – wird u.U. später benötigt)
- API Referenz: https://docs.oracle.com/javase/8/javafx/api/overview-summary.html
- Doku: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm

- **IntelliJ bietet ein Projekt-Template für JavaFX-Applikationen**
  - Menü „File“ → „New Project..“ → links unter Generators „JavaFX“
  - dann rechts „Kotlin“ wählen → „Next“ → „Generate“



JavaFX: Getting Started…

Teil 2: Projektstruktur



IntelliJ generiert viele Ordner und Dateien, die für JavaFX benötigt werden. Nur wenige davon sind für uns relevant.

Anwendungshülle (Einstiegspunkt)

Controller-Klasse (verarbeitet Benutzer-Input)

View Markup (definiert UI-Elemente deklarativ)

JavaFX: Getting Started…

Teil 3: Anwendungshülle / Einstiegspunkt

Benötigte JavaFX-Libraries laden

Von Application-Basisklasse ableiten (vererben)

Methode «start» überschreiben und implementieren

Markup-Datei («View») laden

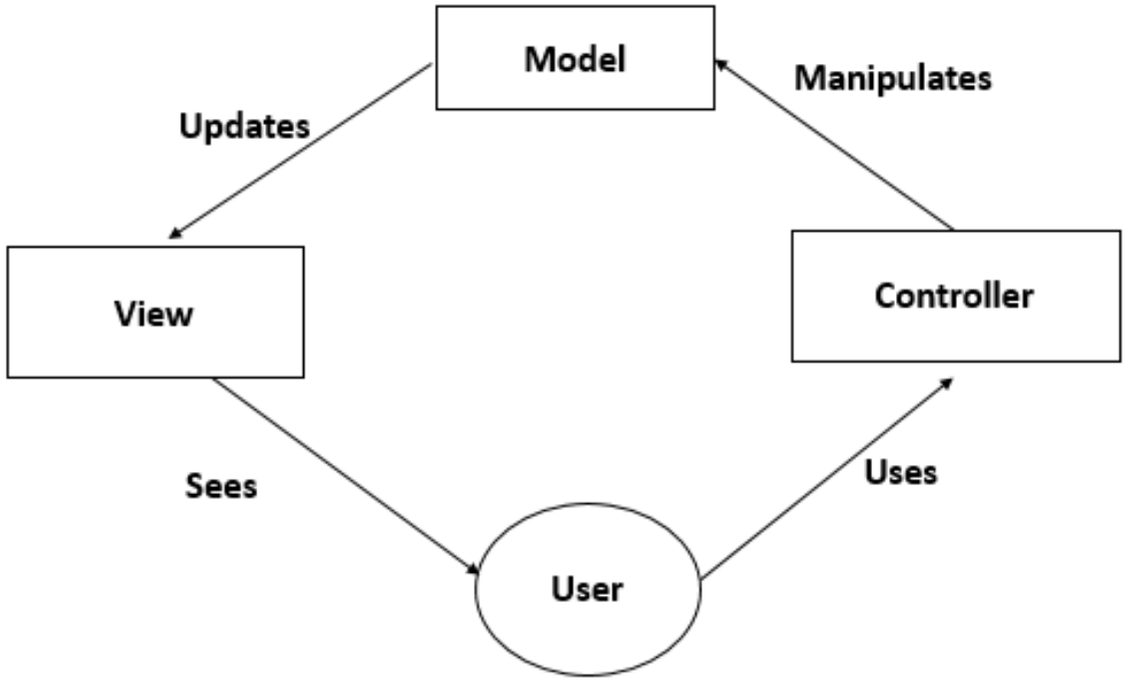
Basis-Einstellungen definieren

Anwendung initialisieren / starten

```
import ...

class HelloApplication : Application() {
    override fun start(stage: Stage) {
        val fxmlLoader = FXMLLoader(HelloApplication::class.java.getResource("hello-view.fxml"))
        val scene = Scene(fxmlLoader.load(), 320.0, 240.0)
        stage.title = "Hello!"
        stage.scene = scene
        stage.show()
    }
}

fun main() {
    Application.launch(HelloApplication::class.java)
}
```

Repetition: «Model-View-Controller» Architektur

„Gewaltentrennung“: System aus drei Teilen mit je eigener Funktion und Daten-Repräsentation



System wird in drei interagierende Teile mit je eigener festgelegter Aufgabe zerlegt. Die erlaubten Interaktions-Richtungen und -Funktionen sind dabei vorgegeben:

- **Model**: Kapselt die Datenschicht
  Interaktion: Aktualisiert den View
  JavaFX: Eine oder mehrere (Daten-) Klasse(n) – optional

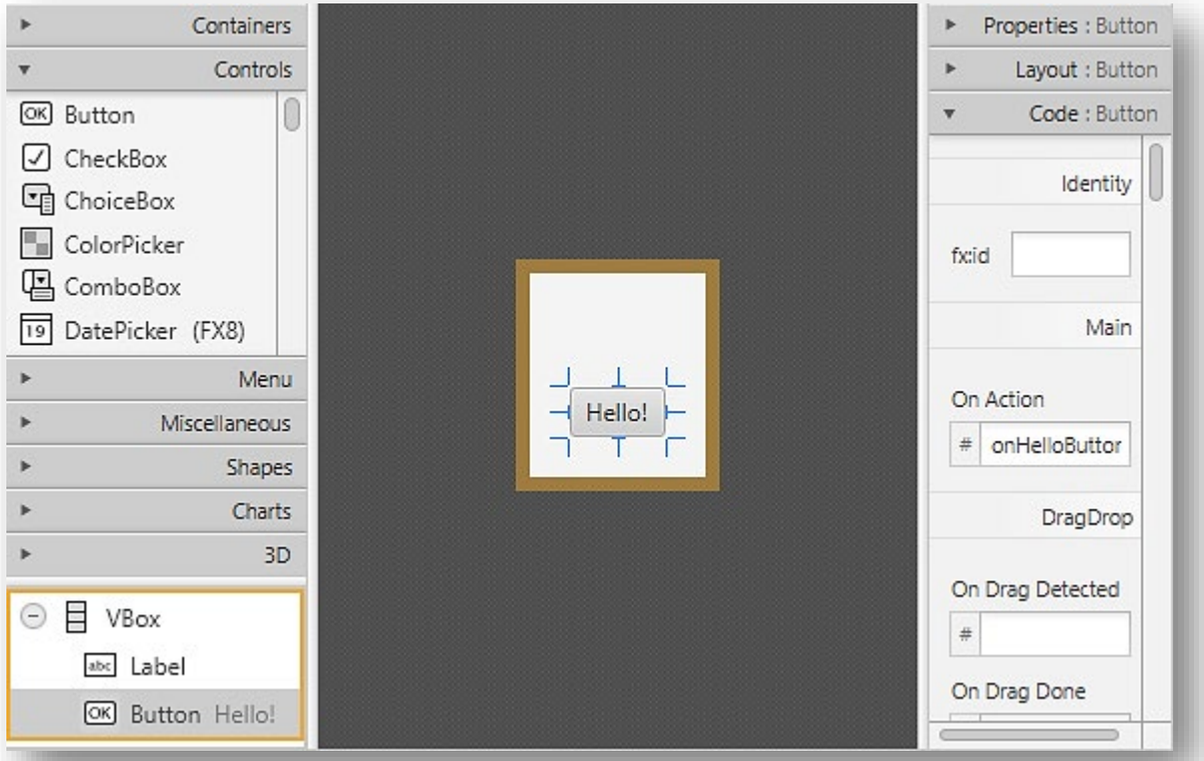
- **Controller**: Behandelt User-Input, steuert Kontrollfluss
  Interaktion: Ändert das Model
  JavaFX: Eine oder mehrere (Logik-) Klasse(n)

- **View**: Stellt die Daten für den Anwender dar (=UI)
  Interaktion: Ruft (indirekt) Controller-Funktionen auf
  JavaFX: Eine oder mehrere FXML Markup-Datei(en)

MVC-View: FXML – Deklarative UI-Entwicklung

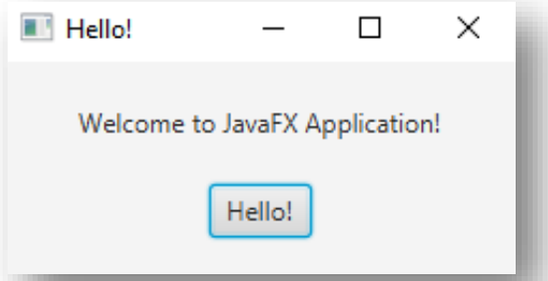
Markup – Scene Builder – Output

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox alignment="CENTER" spacing="20.0„
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.example.demo2.HelloController">

    <padding>
        <Insets bottom="20.0" left="20.0          t="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>

    <Button text="Hello!" onAction="#onHelloButtonClick"/>

</VBox>
```



Zuständige Controller-Klasse

Controller-Methode (EventHandler)



Controller-Klasse

Definiert die Reaktion auf Benutzer-Interaktionen (z.B. Klick auf Button)

```
import ...


class HelloController {
    @FXML
    private lateinit var welcomeText: Label

    @FXML
    private fun onHelloButtonClick() {
        welcomeText.text = "Welcome to JavaFX Application!"
    }
}
```

Benötigte UI-Elemente referenzieren

EventHandler / EventListener implementieren

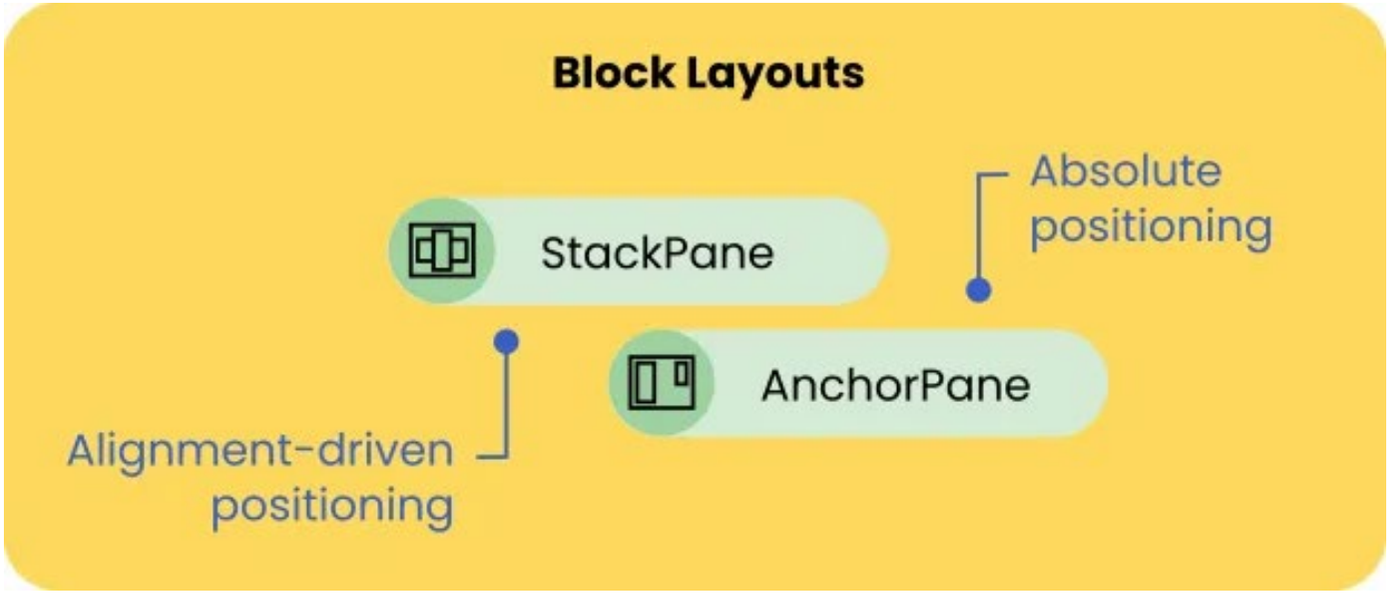
Das Layout-System von JavaFX

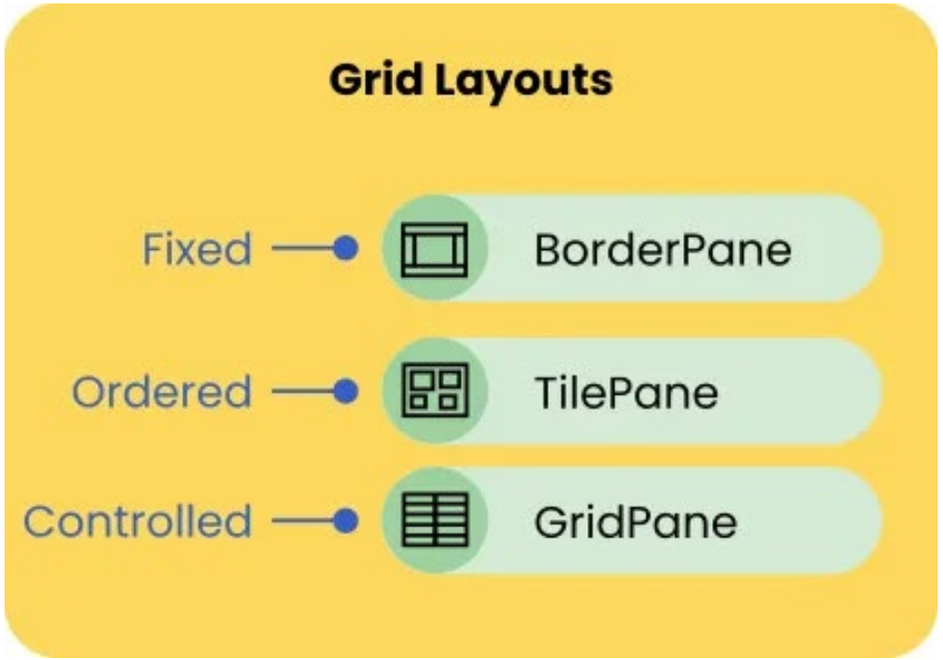
Komplexe reaktive Benutzeroberflächen gestalten

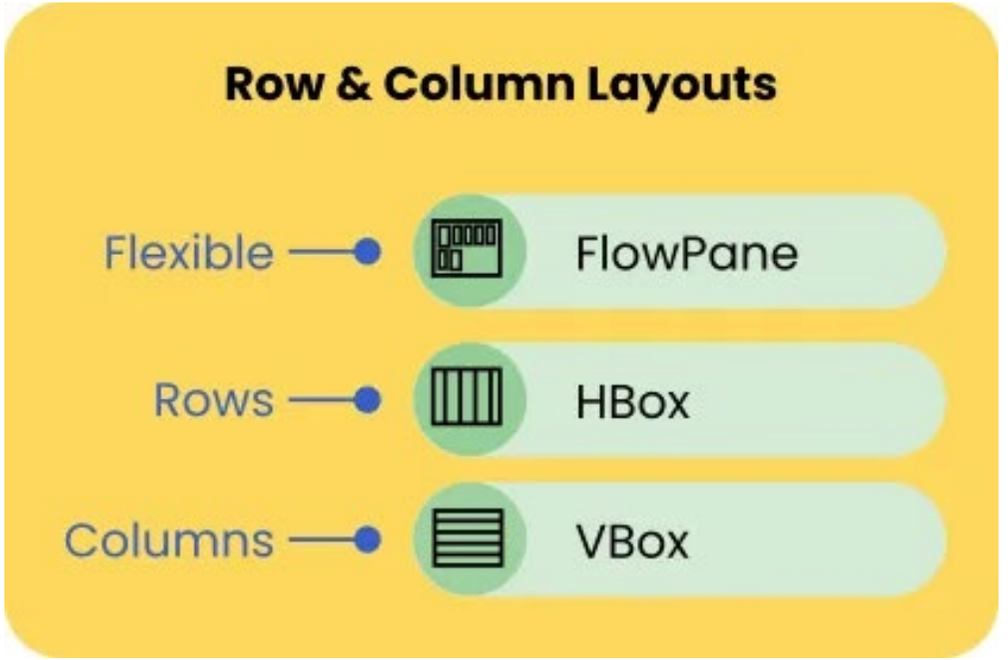
TEKO
SCHWEIZERISCHE FACHSCHULE

- Wie Swing / AWT verwendet JavaFX sog. **LayoutManager** zur Anordnung der UI-Elemente
  - Erleichtert Basis-Design und ermöglicht dessen automatische Anpassung an die Fensterdimensionen
- Layouts können miteinander **kombiniert / ineinander verschachtelt** werden
- Gute Übersicht und Gegenüberstellung der verschiedenen LayoutManagers: https://edencoding.com/javafx-layouts/

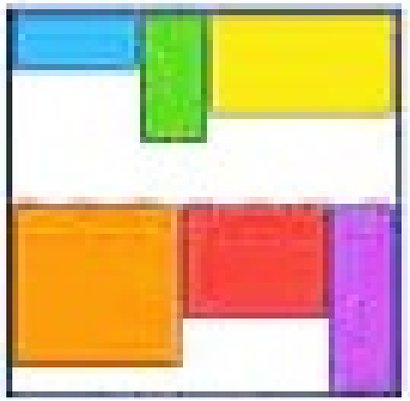
Layouts


Block Layouts
StackPane
Absolute positioning
AnchorPane
Alignment-driven positioning


Grid Layouts
Fixed
BorderPane
Ordered
TilePane
Controlled
GridPane

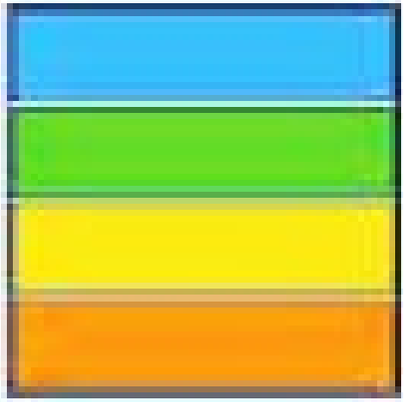

Row & Column Layouts
Flexible
FlowPane
Rows
HBox
Columns
VBox

Row & Column Layouts



FlowPane

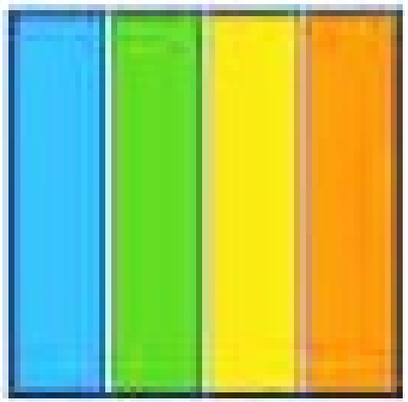
ordnet Elemente der Grösse des Containers entsprechend reaktiv in horizontaler oder vertikaler Flussrichtung Positionierung: Flexibel



VBox

anordnen von Elementen in einer vertikalen Linie / Reihe

Positionierung: Zeilen

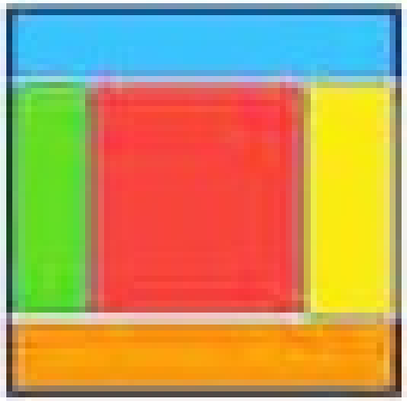


HBox

anordnen von Elementen in einer horizontalen Reihe

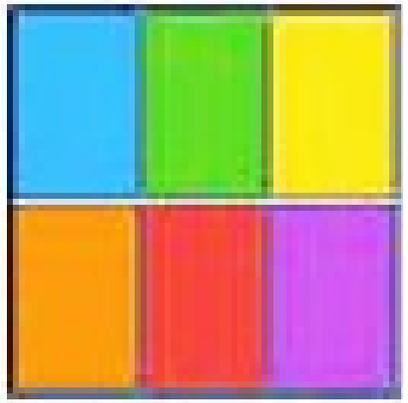
Positionierung: Spalten

Grid Layouts



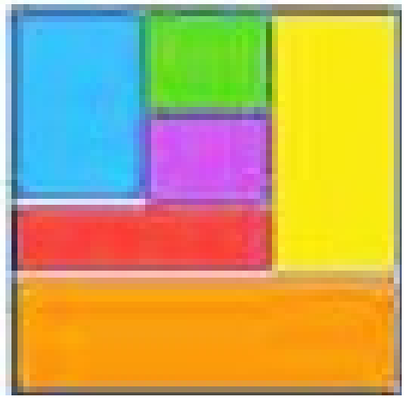
BorderPane

Fünf Bereiche: oben, unten, links, rechts und Mitte. Ideal für Haupt- vs. Nebeninhalte
Positionierung: Fixiert



TilePane

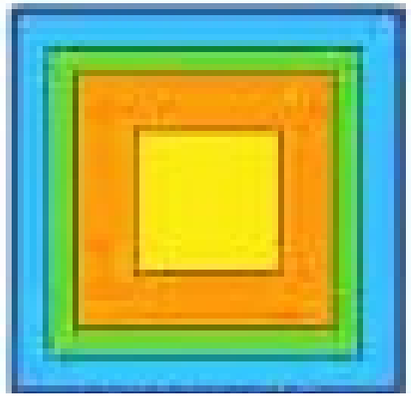
wie FlowPane, ordnet die Kinder jedoch in gleichgrossen Zellen an (Rasteranordnung)
Positionierung: Geordnet



GridPane

Gitter, in dem Elemente in Zeilen und Spalten (Tabelle) angeordnet sind
Positionierung: Kontrolliert

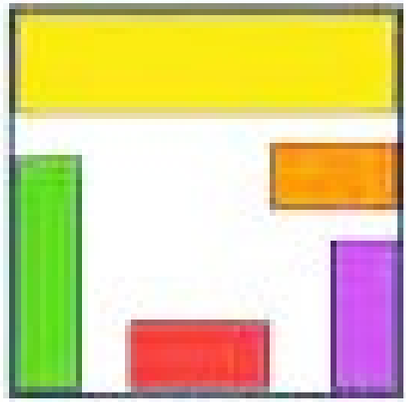
Block Layouts



StackPane

ordnet Elemente übereinander gestapelt an. Elemente bedecken die darunter liegenden.

Positionierung: Ausrichtung



AnchorPane

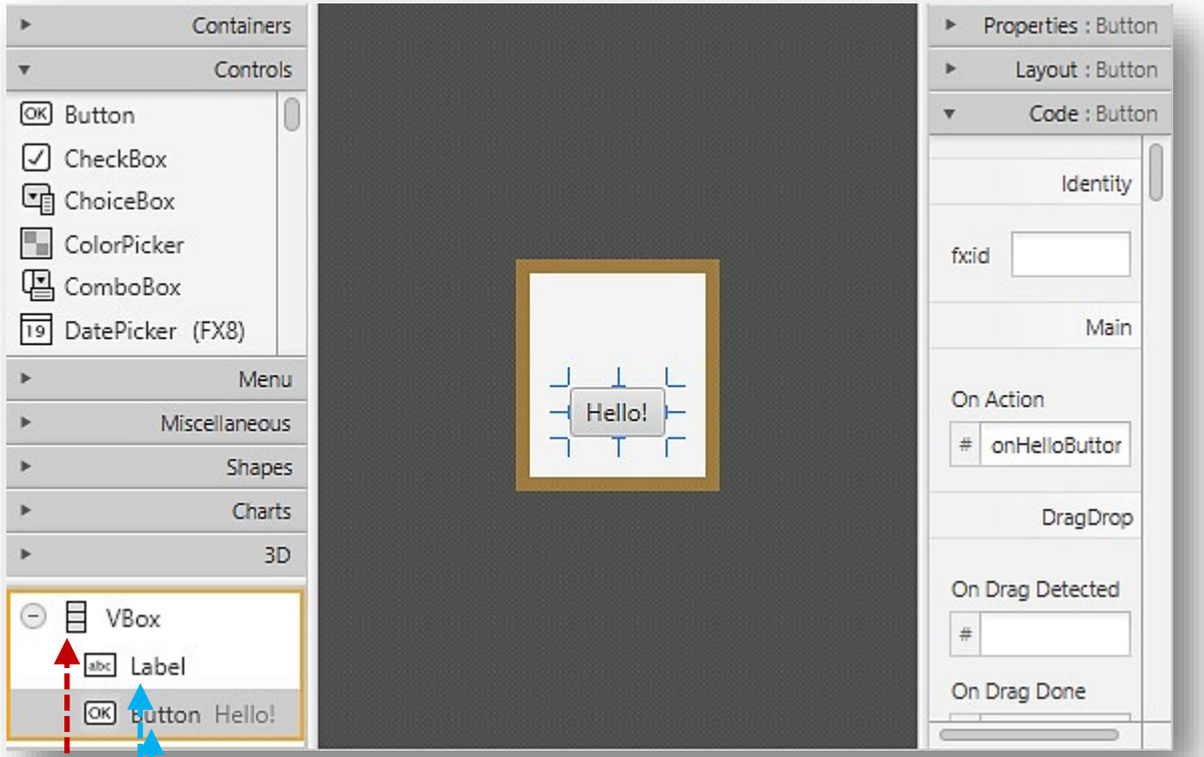
Verankert Elemente an den Ecken oder Seiten des Containers relativ zueinander

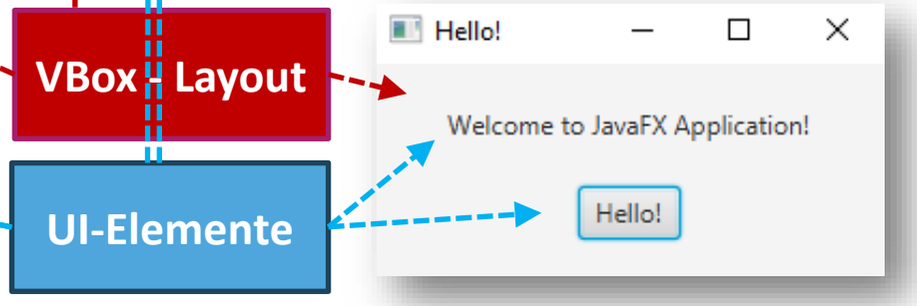
Positionierung: Absolute / relativ

Layout Beispiel

Markup – Scene Builder – Output

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

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>

<VBox alignment="CENTER" spacing="20.0„
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.example.demo2.HelloController">

    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>

    <Button text="Hello!" onAction="#onHelloButtonClick"/>

</VBox>
```





Aufgabe 4.1

« Erste Schritte mit JavaFX »

- Erstellen Sie mit IntelliJ IDEA wie gelernt ein neues JavaFX-Projekt. Führen Sie es aus – installieren Sie bei Bedarf die nötigen Libraries. Holen Sie sich bei Problemen Unterstützung bei Ihren Kolleg:innen.

- Versuchen Sie die Programmstruktur (MVC-Logik) zu verstehen.

- Ändern Sie das Basis-Layout von einer „VBox“ zu einer „HBox“
  - Was müssen Sie alles ändern, damit die Anwendung wieder läuft?
  - Was ist das Ergebnis Ihrer Änderung?

- Falls Sie noch Zeit haben, probieren Sie das Gleiche mit einem anderen Layout, z.B. einer BorderPane, GridPane oder FlowPane statt der VBox.

Layout: Noch mehr Kontrolle

Von den vordefinierten Layouts abweichen

- Alternativ können Elemente auch absolut positioniert werden. Dazu muss als Eltern-Container ein Objekt vom Typ „Pane“ oder „AnchorPane“ verwendet werden:

```
var root = Pane()
var label = Label("Absolut positioniert")
label.setLayoutX(50.0)  // X-Koordinate in Pixeln
label.setLayoutY(100.0) // Y-Koordinate in Pixeln
root.children.add(label)
```

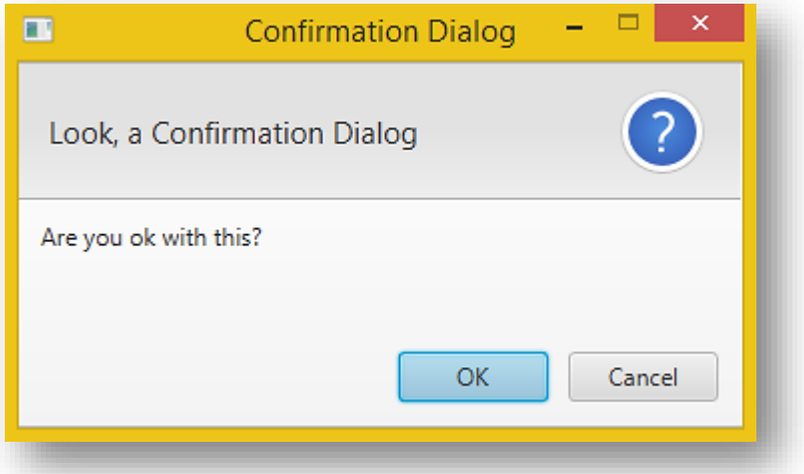
- **ScrollPane** ermöglicht das Scrollen von Inhalten, die grösser als der sichtbare Bereich sind

Dialoge in JavaFX

TEKO
SCHWEIZERISCHE FACHSCHULE

- Wie AWT/Swing verfügt JavaFX über Standard-Dialoge (leicht anders benannt & strukturiert)
- Im Gegensatz dazu lassen sich diese aber wie alle UI Elemente besser anpassen (Styling etc.)

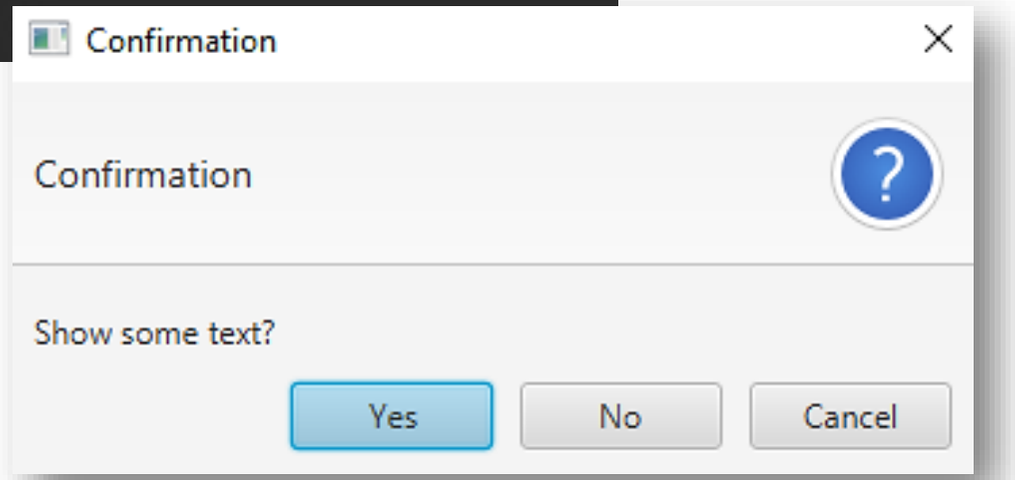
- Folgende Standard UI Elemente sind verfügbar:
  - javafx.scene.control.Dialog
  - javafx.scene.control.Alert
  - javafx.scene.control.TextInputDialog
  - javafx.scene.control.ChoiceDialog



- Dialoge werden oft im Controller definiert (nicht im Markup)
- Gute Einführung & Beispiele: https://code.makery.ch/blog/javafx-dialogs-official/

Alert- / Confirm-Dialog: Beispiel

```
private fun onHelloButtonClick() {

    val alert = Alert(
        AlertType.CONFIRMATION,
        "Show some text?", ButtonType.YES, ButtonType.NO, ButtonType.CANCEL
    )
    alert.showAndWait()

    if (alert.result == ButtonType.YES) {
        welcomeText.text = "Welcome to JavaFX Application!"
    }
}
```



Aufgabe 4.2

« Standard-Dialoge mit JavaFX »

- Implementieren Sie den gezeigten Alert-Dialog in Ihrem JavaFX-Projekt. Holen Sie sich bei Problemen Unterstützung bei Ihren Kolleg:innen.

- Sie haben im letzten Unterricht Standard-Dialoge mit Swing erstellt. Programmieren Sie nun die gleichen Dialoge mit JavaFX (nutzen Sie die erwähnte Einführungsseite als Hilfestellung):
  - Hinweis / Message Box / Information Dialog
  - Eingabe / Text Input Dialog
  - Option / Auswahl / Choice Dialog

- Falls Sie noch Zeit haben, können Sie versuchen, einen Dialog zu „stylen“, z.B. mit einem eigenen Icon, Titel, Farben etc.

JavaFX Styling

TEKO
SCHWEIZERISCHE FACHSCHULE

- Wie AWT / Swing können Farben, Schriftarten, Icons in JavaFX etc. programmiert werden
- Zusätzlich besteht die Möglichkeit, Cascading Style Sheets (CSS) wie in HTML zu verwenden
- Die Benennung der Klassen und Eigenschaften unterscheidet sich jedoch von der in HTML
- Referenzdokumentation:
  https://openjfx.io/javadoc/20/javafx.graphics/javafx/scene/doc-files/cssref.html

- CSS-Dateien müssen im Projekt hinzugefügt und in der start-Methode inkludiert werden:

```
stage.scene.stylesheets.add("ui/base.css")
```

- Details folgen…

Aufgabe 4.3

« JavaFX mit CSS stylen »

- Stylen Sie Ihre Beispielanwendung exemplarisch mit CSS, z.B..
  - Hintergrundfarbe der Anwendung
  - Hinter- oder Vordergrundfarbe eines Buttons
  - Abstände
  - etc.

- Denken Sie daran, Ihre CSS-Datei zu inkludieren
- Studieren Sie die Referenzdokumentation nach Bedarf

JavaFX vs. AWT / Swing

TEKO
SCHWEIZERISCHE FACHSCHULE

JavaFX

Modernere und flexiblere Benutzeroberfläche: Unterstützt CSS, einfache Anpassung des UI

Bessere Integration mit Web-Technologien: Integration von Webinhalten in die Anwendung, Webseiten innerhalb der Anwendung anzeigen

Mobile-Fähigkeit: Zwar keine direkte Integration, aber über Umwege lauffähig auf Android & iOS.

Leistungsstarke Animationen: Unterstützung für Animationen und Effekte erleichtert Erstellung von ansprechenden Benutzeroberflächen

3D-Grafiken: Erstellung von 3D-Grafiken und -Anwendungen, was in Swing nicht so einfach ist

Modernerer Code: Java 8 und höher, MVC, moderne Sprachfeatures wie Lambdas & Streams

AWT / Swing

Einfachheit: Struktur, Logik und Anwendung sind ohne grosse Hürden bzw. Aufwände verständlich

Grössere Benutzerbasis: Lange Geschichte, viele Benutzer, Ressourcen und Bibliotheken verfügbar

Verlässlichkeit: Seit vielen Jahren in zahlreichen Anwendungen eingesetzt. Sehr stabil und bewährt

Plattformunabhängigkeit: Swing-Anwendungen sind plattformunabhängig und laufen auf verschiedenen Betriebssystemen ohne Änderungen am Code

Leichtgewichtiger: Im Vergleich zu JavaFX weniger ressourcenintensiv, von Vorteil insbesondere in eingebetteten Systemen oder auf älteren Hardwareplattformen.

Ausblick: Was noch kommt (oder kommen könnte)…

TEKO
SCHWEIZERISCHE FACHSCHULE

- Weitere Steuerelemente (UI-Elemente) kennenlernen und anwenden
- Komplexe, dynamische Benutzeroberflächen gestalten
- Erweitertes Styling (mit CSS und programmatisch)
- Mehrere Seiten / Views verwenden
- Datenanbindung
- etc.

Weitere Frameworks aus dem Java-Ökosystem

Frameworks – Übersicht

- Grundsätzlich gilt: Ein Framework hat immer einen bestimmten Anwendungszweck
- Entsprechend lassen sich Frameworks in verschiedene Anwendungskategorien einteilen, z.B.

1. **Technische Basis-Anwendungen** zur Unterstützung des Entwicklungsprozesses
2. **Web-, Mobile und Multiplattform-Anwendungen**
3. **Enterprise-Anwendungen** (d.h. grosse, komplexe, langlebige, daten-intensive Applikationen)

– Daneben existieren sehr viele dedizierte Frameworks für fachspezifische Anwendungen (z.B. Medizinalbereich, Vermessung, industrielle Produktion, Sicherheitslösungen etc.)
→ diese sind ausserhalb unseres Fokusses

Frameworks – Technische Basis-Anwendungen

TEKO
SCHWEIZERISCHE FACHSCHULE

- **Maven und Gradle**
  - Build- und Projektverwaltungstools für Java-Anwendungen (inkl. Kotlin).
  - Automatisieren den Build-Prozess und verwalten Abhängigkeiten (Libraries).
  - Häufig auch für die Erstellung und Verwaltung von Java- bzw. Kotlin-Projekten verwendet.

- **Eclipse:**
  - Eine weit verbreitete Java-Entwicklungsumgebung (IDE).
  - Bietet umfangreiche Tools und Plugins zur Entwicklung von Java-Anwendungen.
  - Beliebt bei Entwicklern für die Entwicklung von Java- und anderen Anwendungen.

- **OSGi (Open Service Gateway Initiative)**
  - Ein Modulsystem und Dienstleistungsplattform für Java-Anwendungen.
  - Ermöglicht die Modularisierung und dynamische Aktualisierung von Anwendungen.
  - Häufig in komplexen Systemen und großen Anwendungen verwendet.

- **Hudson / Jenkins**
  - Open-Source Continuous-Integration-Tool.
  - Automatisiert den Build-, Test- und Bereitstellungsprozess.
  - Wird zur Verbesserung der Entwicklungs- und Bereitstellungsprozesse verwendet.

Frameworks – Web, Mobile, Multiplattform-Anwendungen

TEKO SCHWEIZERISCHE FACHSCHULE

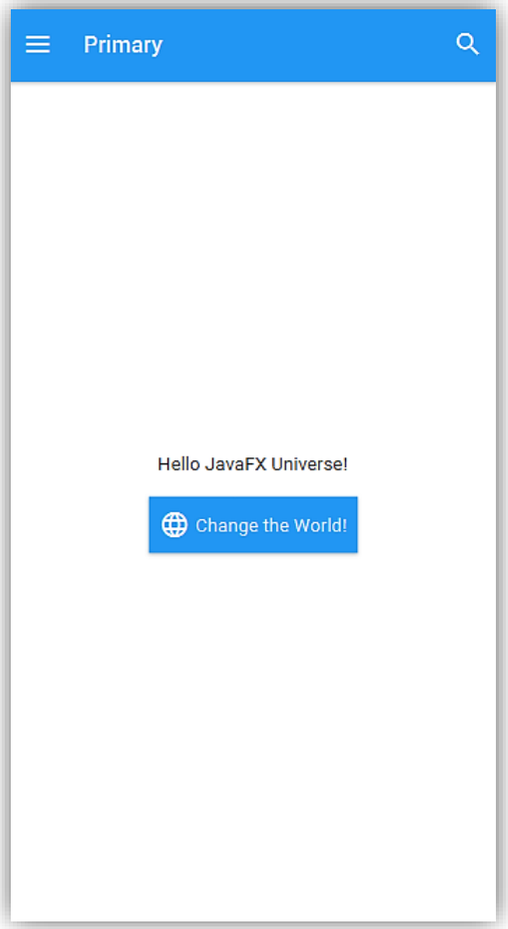
- **JSF (JavaServer Faces)**
  - Ein Java-Framework für die Entwicklung von Webanwendungen.
  - Bietet eine Komponentenbasierte Architektur für die Benutzeroberfläche.
  - Häufig für die Entwicklung von Enterprise-Webanwendungen verwendet.

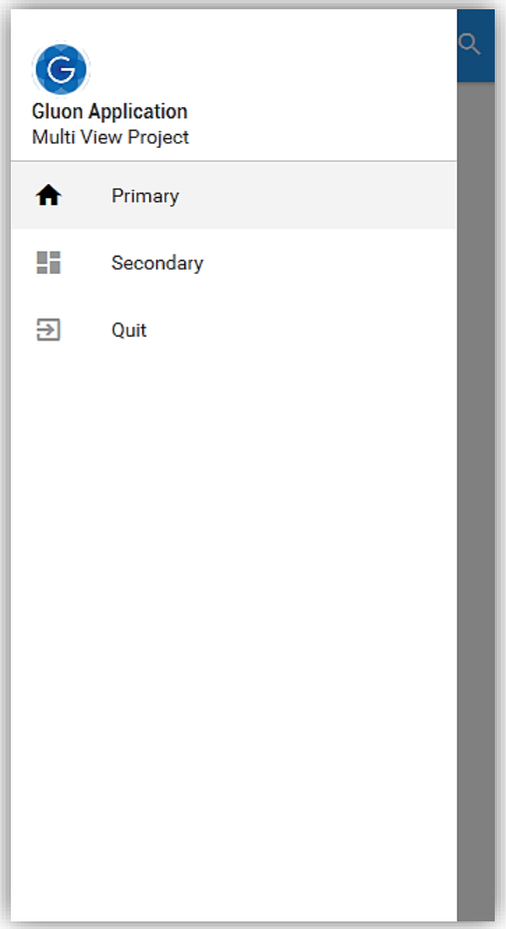
- **LibGDX:**
  - Open-Source-Framework für die Entwicklung von Cross-Platform-Spielen in Java.

- **JavaFX Mobile / Gluon Mobile**
  - JavaFX Mobile war eine Technologie von Oracle für die Entwicklung von mobilen Anwendungen in JavaFX. Allerdings wurde die offizielle Unterstützung für JavaFX Mobile eingestellt, und die Community hat sich auf andere Lösungen verlagert.
  - Nachfolger: Mithilfe des Gluon-Frameworks bzw. JavaFXPorts lassen sich Apps für Android und iOS entwickeln.

- **Codename One**
  - Plattformübergreifendes Framework für die Entwicklung von mobilen Anwendungen in Java / Kotlin.
  - Entwicklung von nativen Apps, die auf iOS, Android, Windows Phone und anderen Plattformen laufen („Cross Compiling“)
  - Plattform-spezifische Apps können lokal oder mittels Cloud-Dienst gebaut werden, sodass keine Hardware erforderlich ist

Gluon Mobile: Beispiel






```java
public class GluonApplication extends Application {

    public static final String PRIMARY_VIEW = HOME_VIEW;
    public static final String SECONDARY_VIEW = "Secondary View";

    private final AppManager appManager = AppManager.initialize(this::postInit);

    @Override
    public void init() {
        appManager.addViewFactory(PRIMARY_VIEW, () -> new PrimaryView().getView());
        appManager.addViewFactory(SECONDARY_VIEW, () -> new SecondaryView().getView());

        DrawerManager.buildDrawer(appManager);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        appManager.start(primaryStage);
    }

    private void postInit(Scene scene) {
        Swatch.BLUE.assignTo(scene);

        scene.getStylesheets().add(
                    GluonApplication.class.getResource("style.css").toExternalForm());
        scene.getWindow().setWidth(400.0);
        scene.getWindow().setHeight(800.0);
        ((Stage) scene.getWindow()).getIcons().add(
                    new Image(GluonApplication.class.getResourceAsStream("/icon.png")));
    }

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

https://docs.gluonhq.com / https://gluonhq.com/products/mobile/javafxports/

Frameworks – Enterprise-Anwendungen

- **Jakarta EE (früher J2EE / JEE, Java Enterprise Edition)**
  - Umfangreiche Plattform für die Entwicklung von Unternehmensanwendungen.
  - Enthält verschiedene Spezifikationen wie EJB, JPA und Servlets.
  - Bietet eine umfangreiche Palette von Diensten für Enterprise-Anwendungen.

- **EJB (Enterprise JavaBeans)**
  - Eine Spezifikation für die Entwicklung von Enterprise-Anwendungen und Datenbank-Integrationen.
  - Ermöglicht die Erstellung wiederverwendbarer Geschäftslogikkomponenten.
  - Häufig in JEE-Anwendungen verwendet.

- **JPA (Java Persistence API)**
  - Eine Spezifikation für das Persistieren von Daten in Java-Anwendungen.
  - Ermöglicht die objektorientierte Datenbankinteraktion.
  - Häufig in Kombination mit ORM-Frameworks wie Hibernate verwendet.

- **Spring**
  - Mächtiges Framework für die Entwicklung von Enterprise-Anwendungen. Aktuell DAS grosse Framework im Java-Bereich
  - Verfolgt moderne Kernkonzepte wie Inversion of Control / Dependency Injection und aspektorientierte Programmierung
  - Modular aufgebaut: Core Container für IoC/DI, Data Access für Datenzugriff, MVC für Webanwendungen, Security
  - Spring Boot: Erweiterung von Spring, welche die Entwicklung einfacher machen soll, nach „Best Practices & Patterns“ aufgebaut