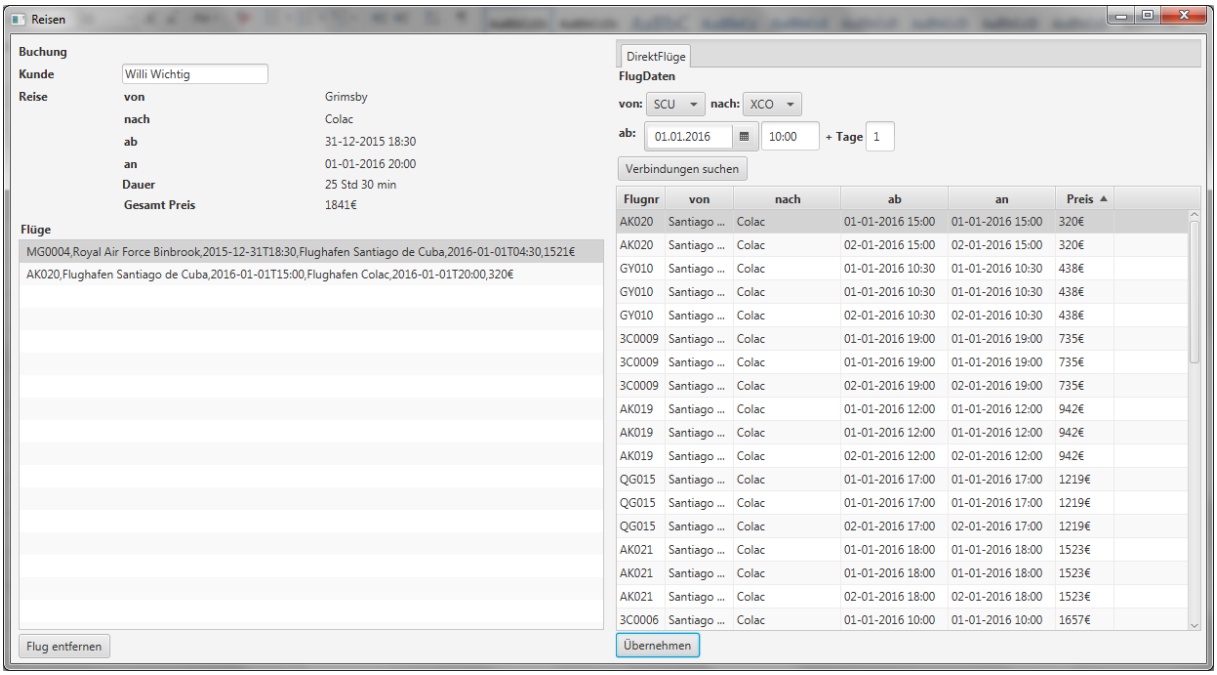
Aufgabe 4 Ein JavaFX GUI für ein sehr einfaches Reisebuchungssystem

Aufgabe

Entwickeln Sie eine JavaFX GUI für das Buchen von Rundreisen, die nur aus Direktflügen bestehen. Verwenden Sie die Lösungen aus Aufgabe 2 und Aufgabe 3, indem Sie die Projekte in ihr GUI Projekt einbinden.

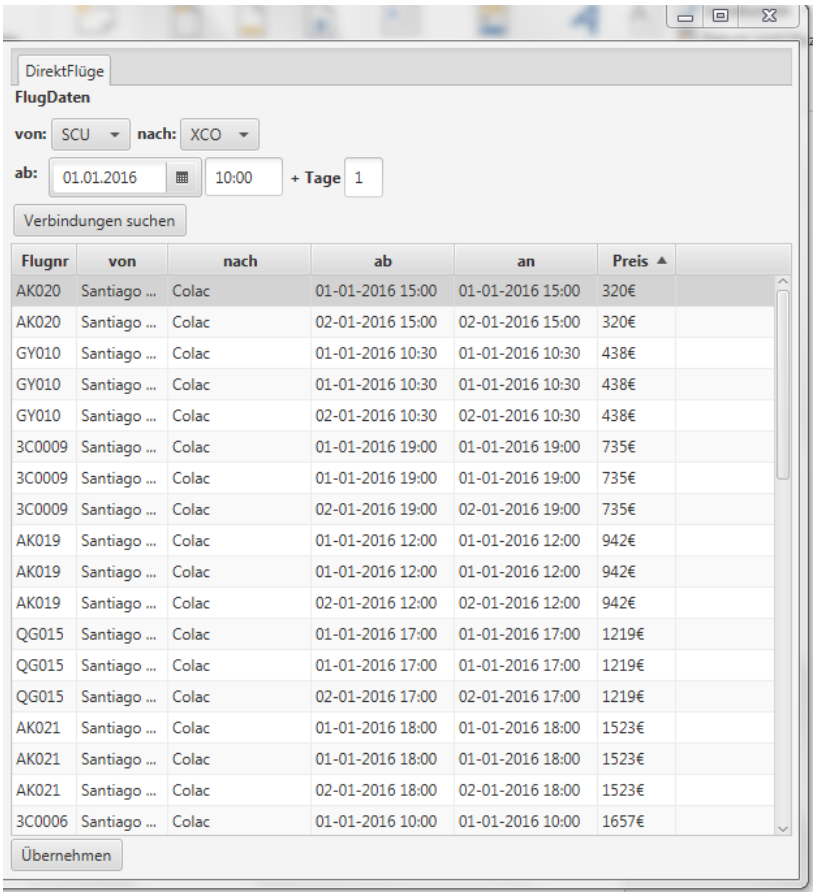
Verwenden Sie für das Erstellen der statischen View-Bestandteile unbedingt den SceneBuilder, wenn Sie das erste Mal eine GUI entwickeln.

Zum Aussehen des GUI's:



Ihre GUI teilt sich in eine linke Seite, die die gebuchten Flüge und die Gesamtübersicht für die Reise angezeigt, und eine rechte Seite, in der nach möglichen Flügen gesucht werden kann.

Zum Aufbau des rechten Teils für die Suche nach Flügen



Flugnr	von	nach	ab	an	Preis
AK020	Santiago ...	Colac	01-01-2016 15:00	01-01-2016 15:00	320€
AK020	Santiago ...	Colac	02-01-2016 15:00	02-01-2016 15:00	320€
GY010	Santiago ...	Colac	01-01-2016 10:30	01-01-2016 10:30	438€
GY010	Santiago ...	Colac	01-01-2016 10:30	01-01-2016 10:30	438€
GY010	Santiago ...	Colac	02-01-2016 10:30	02-01-2016 10:30	438€
3C0009	Santiago ...	Colac	01-01-2016 19:00	01-01-2016 19:00	735€
3C0009	Santiago ...	Colac	01-01-2016 19:00	01-01-2016 19:00	735€
3C0009	Santiago ...	Colac	02-01-2016 19:00	02-01-2016 19:00	735€
AK019	Santiago ...	Colac	01-01-2016 12:00	01-01-2016 12:00	942€
AK019	Santiago ...	Colac	01-01-2016 12:00	01-01-2016 12:00	942€
AK019	Santiago ...	Colac	02-01-2016 12:00	02-01-2016 12:00	942€
QG015	Santiago ...	Colac	01-01-2016 17:00	01-01-2016 17:00	1219€
QG015	Santiago ...	Colac	01-01-2016 17:00	01-01-2016 17:00	1219€
QG015	Santiago ...	Colac	02-01-2016 17:00	02-01-2016 17:00	1219€
AK021	Santiago ...	Colac	01-01-2016 18:00	01-01-2016 18:00	1523€
AK021	Santiago ...	Colac	01-01-2016 18:00	01-01-2016 18:00	1523€
AK021	Santiago ...	Colac	02-01-2016 18:00	02-01-2016 18:00	1523€
3C0006	Santiago ...	Colac	01-01-2016 10:00	01-01-2016 10:00	1657€

Im oberen Bereich befinden sich 2 Auswahlboxen, in denen der Abflughafen und Ankunfts-Flughafen ausgewählt werden können. In der Darstellung sollen nur die Iata-Codes sichtbar sein. Ändern Sie dazu die ***toString*** Methode von ***IataAirport***. Verbinden Sie jede Auswahlbox mit einem Tooltip, der den Namen des Flughafens anzeigt. Listen von ***IataAirport*** Objekten können Sie über eine Methode der Klasse ***FlugGenerator*** erzeugen (siehe Hilfestellungen).

In der dritten Zeile befindet sich ein ***DatePicker*** zur Auswahl des Abflugdatums und ein Textfeld für die Eingabe von Uhrzeiten, sowie ein Textfeld um das Suchintervall um + Tage zu erhöhen. Stellen Sie den ***DatePicker*** so ein, dass nur der aktuelle Tag auswählbar ist. Das Textfeld darf nur gültige Zeitangaben akzeptieren (siehe Hilfestellungen). Das +Tage Feld nur ganze Zahlen.

Mit dem Button *„Verbindung Suchen“* werden Direktflüge für gewählten Zeitraum in der Tabelle angezeigt. Eine Flugliste können Sie sich mit einer Methode der Klasse ***FlugGenerator*** erzeugen (siehe Hilfestellungen).

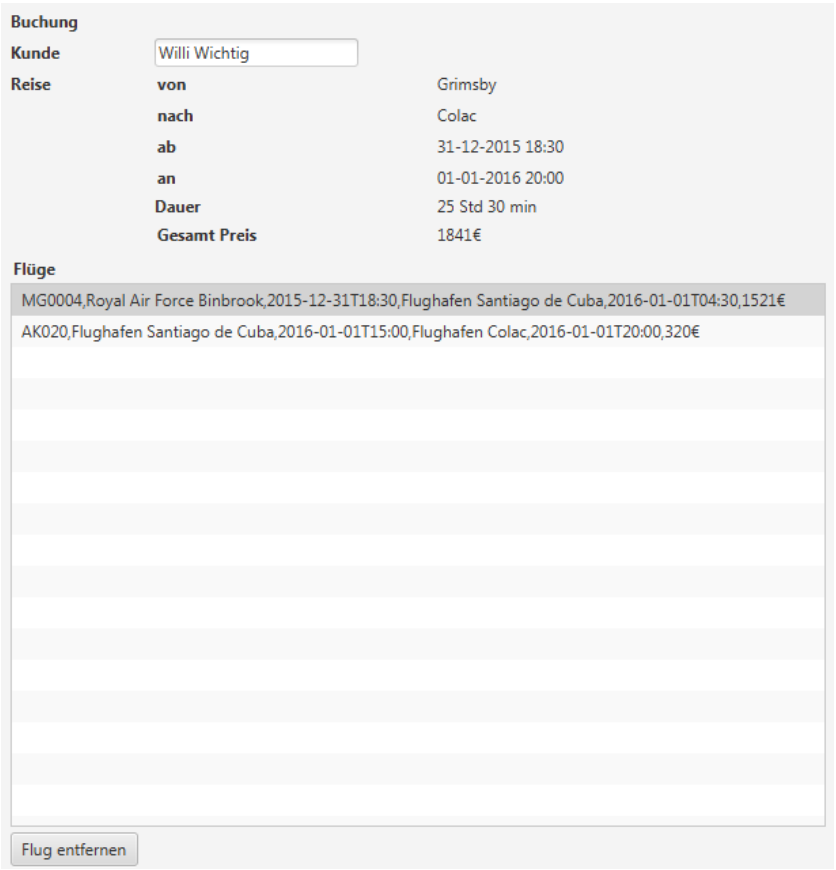
Die Spalten *Flugnr*, *von*, *nach*, *ab*, *an* und *Preis* sollen mit den entsprechenden Eigenschaften der Direktflüge verknüpft werden. Damit dazu nicht die Originalklasse ***DirektFlug*** geändert werden muss, packen wir die ***DirektFlug***-Objekte in ***DirektFlugWrapper***-Objekte ein (siehe Konstruktor der Klasse DirektFlugWrapper).

Die Klasse ***DirektFlugWrapper*** wird mitgeliefert und ist so präpariert, dass die Properties mit den Spalten der Tabelle einfach verknüpft werden können (wie das geht steht im Script).

Mit dem Button „*Übernehmen*“ wird die aktuell selektierte Tabellenzeile (das Objekt, das sich dahinter verbirgt) in die Flugliste auf der linke Seite des GUI’s übertragen. Ist keine Zeile selektiert, passiert nichts, insbesondere auch kein Fehler!

Der Flug, der in die Buchung auf der linken Seite übertragen wird, muss aus der Tabelle gelöscht werden!

Zum Aufbau des linken Teils für die Ansicht der Reisebuchung



Im oberen Teil befinden sich die allgemeinen und akkumulierten Daten der Buchung. Im unteren Teil eine Liste der gewählten Flüge.

Der Name des Kunden soll frei in das Textfeld eingetragen werden können.

Alle Daten der Rundreise müssen abhängig von den enthaltenen Flügen berechnet werden. D.h. Immer, wenn ein Flug zur Liste hinzugefügt oder entfernt wird, müssen die Werte *von, nach, ab, an, Dauer, Gesamt Preis* aktualisiert werden. Realisieren Sie dies, indem Sie die Text-Properties der entsprechenden Labels/Textfelder der GUI an die Properties der Klasse ***RundReiseWrapper*** binden

und beim Hinzufügen / Löschen von Elementen in der GUI-Liste, die ***add***/***remove*** Methoden der Klasse ***RundReiseWrapper*** nutzen. Die Wrapperobjekte werden mit einer Referenz auf eine ***RundReise*** erzeugt, so dass die Funktionalität auf einen Rundreise zurückgeführt werden kann.

Die Klasse ***RundReiseWrapper*** wird mitgeliefert und ist bereits mit den notwendigen Properties ausgestattet. Für die Eigenschaft „***Dauer***“ können Sie nicht mit einem Property-Binding arbeiten. Hier müssen Sie einen ***ChangeListener*** auf die Property der Klasse ***RundReiseWrapper*** registrieren, der dann ein Duration-Objekt geeignet für die Ausgabe formatiert (siehe Hilfestellungen).

Mit dem Button „*Flug entfernen*“ wird der Flug aus der Liste gelöscht und wieder in die Tabelle der Flüge rechts eingetragen.

Hilfestellungen und mitgelieferte Klassen

Klasse FlugGenerator

Methode: `getAirports()` Liefert eine Liste mit IataAirport-Objekten

Methode:
```
generateDirektFlugListe(LocalDateTime start,
                        int plusDays, IataAirport von, IataAirport nach)
```

erzeugt ab dem Startdatum *start* für die Strecke zwischen den Flughäfen *von*, *nach* Direktflüge für die Anzahl Tage *plusDays +1.* Das Ergebnis ist eine Liste von Direktflügen.

Konstruktor: `FlugGenerator()`: Erzeugt eine Liste von 20 zufällig gewählten `IataAirport` und eine Liste von 5 zufällig gewählten `IataAirline`-Objekten.

Konstruktor: `FlugGenerator(int numAirports, int numAirlines)`: siehe Defaultkonstruktor

Textfelder mit Format- kontrollierter Eingabe

Textfelder können über ***TextFormatter*** gültige Eingaben kontrollieren. Der Formatierer verwendet für die Prüfung einer gültigen Eingabe einen Subtyp von ***StringConverter***, der einen String gemäß eines Formats parsed. Ist die Eingabe nicht gültig, dann wird diese nicht in das Textfeld übernommen.

In der JavaFX GUI Bibliothek gibt es für viele Klassen bereits fertige Implementierungen für diese Konvertierer. Für ***LocalTime*** ist das der ***LocalTimeStringConverter.*** Das Argument ***FormatStyle***.***SHORT*** besagt, dass die Zeit im Format *HH:mm* eingegeben werden soll.

```
flugZeitAb.setTextFormatter(new TextFormatter<LocalTime>(
                            new LocalTimeStringConverter(FormatStyle.SHORT)));
```

Duration Objekt für die Ausgabe formatieren

```
String.format("%d Std %d min",
       rrw.getDauer().toHours(),
       rrw.getDauer().minusHours(rrw.getDauer().toHours())
                                                    .toMinutes()))
```

Hier steht ***rrw*** für ein Objekt von Typ ***RundReiseWrapper***.

Klasse DirektFlugWrapper

```
public class DirektFlugWrapper {

    private DirektFlug direktFlug;
    private SimpleStringProperty vonProperty;
    private SimpleStringProperty nachProperty;
    private SimpleStringProperty abProperty;
    private SimpleStringProperty anProperty;
    private SimpleStringProperty flugNrProperty;
    private SimpleObjectProperty<GeldBetrag> preisProperty;

    public static List<DirektFlugWrapper> wrap(List<DirektFlug> dfl) {
        return dfl.stream().map(df -> new DirektFlugWrapper(df)).collect(Collectors.toList());
    }

    public DirektFlugWrapper(DirektFlug direktFlug) {
        this.direktFlug = direktFlug;
        this.flugNrProperty = new SimpleStringProperty(direktFlug.getFlugNummer().toString());
        this.vonProperty = new SimpleStringProperty(direktFlug.getStartOrt().toString());
        this.nachProperty = new SimpleStringProperty(direktFlug.getEndOrt().toString());
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
        this.abProperty = new SimpleStringProperty(direktFlug.getBeginn().format(formatter));
        this.anProperty = new SimpleStringProperty(direktFlug.getEnde().format(formatter));
        this.preisProperty = new SimpleObjectProperty<GeldBetrag>(direktFlug.getPreis());
    }
```

Klasse RundReiseWrapper

```
public class RundReiseWrapper {

    private SimpleStringProperty vonProperty;
    private SimpleStringProperty nachProperty;
    private SimpleStringProperty abProperty;
    private SimpleStringProperty anProperty;
    private SimpleObjectProperty<GeldBetrag> preisProperty;
    private RundReise reise;
    private SimpleObjectProperty<Duration> dauerProperty;

    public RundReiseWrapper(RundReise reise) {
        this.reise = reise;
        this.vonProperty = new SimpleStringProperty(
                (reise.getStartOrt() == null) ? "" : reise.getStartOrt()
                        .toString());
        this.nachProperty = new SimpleStringProperty(
                reise.getEndOrt() == null ? "" : reise.getEndOrt().toString());
        DateTimeFormatter formatter = DateTimeFormatter
                .ofPattern("dd-MM-yyyy HH:mm");
        this.abProperty = new SimpleStringProperty(
                reise.getBeginn() == null ? "" : reise.getBeginn().format(
                        formatter));
        this.anProperty = new SimpleStringProperty(reise.getEnde() == null ? ""
                : reise.getEnde().format(formatter));
        this.dauerProperty = new SimpleObjectProperty<Duration>(
                reise.getDauer()==null? Duration.ofMinutes(0) : reise.getDauer());
        this.preisProperty = new SimpleObjectProperty<GeldBetrag>(
                reise.getPreis() == null ? new Euro(0) : reise.getPreis());
    }
```

Wichtige Ergänzung zu Aufgabe 3:

Damit die Methode ***generateDirektFlugListe*** der Klasse ***FlugGenerator*** arbeiten kann, müssen Sie in den Klassen für die Flüge folgende Änderungen vornehmen.

```
public abstract class Flug extends ReiseBaustein {
	public abstract Flug delayDays(int numDays);
}


public abstract class OneWayFlug extends Flug {
	@Override
	public abstract OneWayFlug delayDays(int numDays);

}


public class DirektFlug extends OneWayFlug {
	@Override
	public DirektFlug delayDays(int numDays) {
		return new DirektFlug(flugNummer, anbieter, start, ziel,
		abflugZeit.plusDays(numDays),ankunftsZeit.plusDays(numDays), preis);
	}
```

```
public class StopOverFlug extends OneWayFlug {
	@Override
	public StopOverFlug delayDays(int numDays) {
		StopOverFlug thisDelayed = new StopOverFlug();
		fluege.stream().forEach(flug ->
			thisDelayed.add(flug.delayDays(numDays)));
		return thisDelayed;
	}

public class ToFroFlug extends Flug {
	@Override
	public Flug delayDays(int numDays) {
		return new ToFroFlug(anbieter, hinFlug.delayDays(numDays),
					rueckFlug.delayDays(numDays));
	}
```