Module 4 Assignment

Description

In this assignment, you will create a complete application for editing airline flight information, including a file-based database, so that its data is persistent between runs.

First Steps

1. Switch Branches:

- Using the command line, navigate to your solutions directory.
- Switch to the `main` branch and merge in your `module3` branch.
- Create (and switch to) a new branch named `module4`.

2. Create IntelliJ Project:

- Create a new IntelliJ project using the JavaFX template.
- Name the project `module4` and place it inside your solutions directory.
- IMPORTANT: Disable the 'create git repository' checkbox since you've already created a repository.
- The JavaFX template creates several files (two `.java` files and an `.fxml` file). As this project proceeds, rename these files to something more appropriate for this assignment. For example, you could rename `HelloApplication.java` to `FlightScheduleApplication.java`.

Domain Model

The majority of a major airline's business results from recurring flights. For example, Delta Airlines flight DL1331 flies from Pittsburgh (PIT) to Atlanta (ATL), departing Pittsburgh every day at 1:30 PM (Eastern Time) and arriving in Atlanta at 3:00 PM (Eastern Time). The same flight also travels from ATL to PIT, departing at 11:00 AM and arriving at 12:30 PM. All times here are 'scheduled time', and the actual flight times may vary significantly. A 'flight designator' may correspond to several flights on a given day, and it is possible for those flights to be in the air simultaneously (if one is late, for example).

This is further complicated by the fact that the same flight might have multiple stops. However, we will not handle that case in our simple application.

We need a database to track the scheduled times of a given flight designator. We don't care about actual calendar days, just days of the week. That is, we only need to

know if flight DL1331 generally flies on Mondays, for instance. Our model object looks like this:

ScheduledFlight:

The `ScheduledFlight` class should include the following fields:

- flightDesignator
- departureAirportIdent (e.g., the IATA code for Pittsburgh International is PIT, but its ident is KPIT)
- departureTime (use `java.time.LocalTime` for this field)
- arrivalAirportIdent
- arrivalTime (use `java.time.LocalTime` for this field)
- daysOfWeek (use a `HashSet` to hold the days of the week that this flight is scheduled)

Create this class with getters and setters for all fields. Include rudimentary validation in your setters: if the setter is called with `null`, throw `InvalidArgumentException`. It's recommended to write unit tests to verify that your class is working properly.

Database

Create an `AirlineDatabase` class that includes the following methods:

- `getScheduledFlights()`
- `addScheduledFlight(ScheduledFlight sf)`
- `removeScheduledFlight(ScheduledFlight sf)`
- `updateScheduledFlight(ScheduledFlight sf)`

Implement these methods similarly to the way demonstrated in the lecture videos.

Create an `AirlineDatabaseIO` class, following the pattern shown in the lecture videos:

- `static save(AirlineDatabase ad, OutputStream strm)`
- `static load(InputStream strm)` — returns an `AirlineDatabase`

Again, writing unit tests is recommended to verify that your classes are working properly.

User Interface

Create a table-based user interface, including a separate (not in-table) editor similar to the one created for Customers in the lecture videos. In the table display:

- flightDesignator
- departureAirportIdent
- arrivalAirportIdent
- Days of week as a string with one character per day of the week that this flight includes (use 'R' for Thursday and 'U' for Sunday)

In the detail editor, include `TextField` inputs for all fields except the day of the week. For the day of the week, use JavaFX toggle buttons, one for each day.

You may separate the table and detail editors into their own components, like in the lecture, or create one large application. Either approach is fine, but following the lecture example is recommended.

Submitting

Commit and push your work one last time, then verify that your files are correctly displayed on GitHub (ensure you're on the correct branch when checking GitHub). Next, submit this assignment on Canvas by pasting the URL of your GitHub repository into the supplied input. This will be the same URL that you submitted in Module 1. Note that Canvas may try to display a preview of the repository, but if the repository is private, it may show an error. As long as your URL is correct, everything is fine.