TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr. Max Mühlhäuser

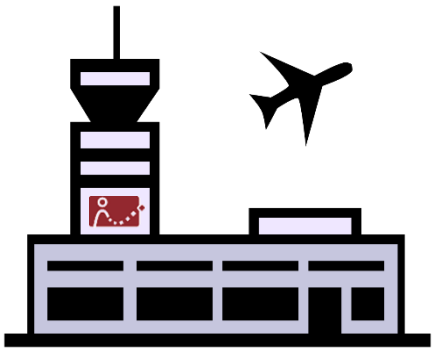
TELEKOOPERATION
Fachbereich Informatik
Hochschulstr. 10
64289 Darmstadt

TK1: Distributed Systems – Programming & Algorithms

1st Programming Exercise
Submission Date: 25.11.2020

By handing in a solution you confirm that you are the exclusive author(s) of all the materials.

Introduction



This year, the practical exercises will be all about the virtual TK Airport (FRA). With two terminals and 64.5 million passengers every year, the TK Airport is one of the world's largest virtual airport. Due to the increasing number of virtual passengers, the airport needs to modernize its IT-infrastructure.

Your task throughout the TK1 practical exercises is to develop applications that help the airport to deal with the next century challenges using the methodologies introduced in the lecture.

P1. Arrival and Departure Scheduling

The first application that needs urgent modernization is the arrival and departure scheduling. Each day hundreds of aircrafts arrive or depart the virtual TK airport. For handling all these arrivals and departures, a central platform is required that enables the airport to schedule all flights.

Your Task

Your task is to develop a client / server application using **Java RMI** that enables the airport to maintain a central list of arrivals and departures (flights) of all aircrafts.

On a GUI, the clients should be able to create new flights, edit or delete existing flights. Multiple clients should be able to access the list at the same time and changes to the schedule should refresh all other clients.

Each flight contains the following fields (marked bold):

- Operating Airline
  - **IATA Code** (e.g., LH)
  - **Name** (e.g., Lufthansa)
- Aircraft Type
  - **Model name** (e.g., A380)
- **Flight number** (with IATA Code unique for the day, e.g., 591)
- **Departure Airport** (e.g., NBO)
- **Arrival Airport** (e.g., FRA)
- **Origin Date** (e.g., 2018-08-13)
- Arrival (if it is an arrival flight)
  - **Scheduled date and time** (e.g., 2018-08-13T03:40:00Z)
  - **Terminal** (e.g., 1)
  - **List of gates** (e.g., C15A)
  - **Estimated date and time** (e.g., 2018-08-13T03:20:00Z)
- Departure (if it is an departure flight)
  - **Scheduled date and time** (e.g., 2018-08-13T20:25:00Z)
  - **Terminal** (e.g., 1)
  - **List of gates** (e.g., B54)
  - Check-in Information
    - **Check-in location** (e.g., 1)
    - **Check-in counter** (e.g., from 664 to 673)
    - **Check-in date and time** (e.g., from 2018-08-13T17:55:00Z to 2018-08-17T19:45:00Z)
- **Flight status**
  - 'B' = Arrival by bus at Concourse B
  - 'D' = Diverted
  - 'I' = Undefined late arrival or departure
  - 'L' = Aborted departure
  - 'M' = Flight delayed until tomorrow

- 'S' = Definitively canceled flight
- 'X' = Canceled flight for which there may be a replacement
- 'Y' = Return to stand
- 'Z' = Returned to apron

Task Requirements

The following requirements must be fulfilled for this practical exercise:

Server

- The server maintains a list of all flights (there is no need to a database, a list in memory is sufficient).
- The server distributes all changes to all logged-in clients (login of a client, updated flight data).
- After the client has logged in at the server, the server will send a list of all flights to the client.
- No GUI for server needed.

Client

- The GUI for the clients should be Swing or JavaFX-based (*recommendation for simplification*: use the Eclipse WindowBuilder which allows to design a simple UI via drag-and-drop)
  - The GUI shows a list of all flights with at least the operating airline, the flight number, departure or arrival, time of arrival or departure and the terminal.
  - The list is updated whenever a flight has been modified (i.e., if another client changes a flight).
  - The user can create new flights (with the above-mentioned fields), edit or deleting existing flights. A detailed view of the flights allows inspecting all fields.
- Each client has a unique name (e.g., the username or random string) which is used for identification.
- The client sends all changes of a flight to the server.
- The client should load the stubs from the Java RMI registry.

General

- Please generate a working Gradle project which contains two run-tasks (server start and client start). We will not evaluate your handed in solution without the Gradle script.
- Please test your packaged project and reimporting it into an empty eclipse workspace.
- You can assume that the Java RMI registry is started.

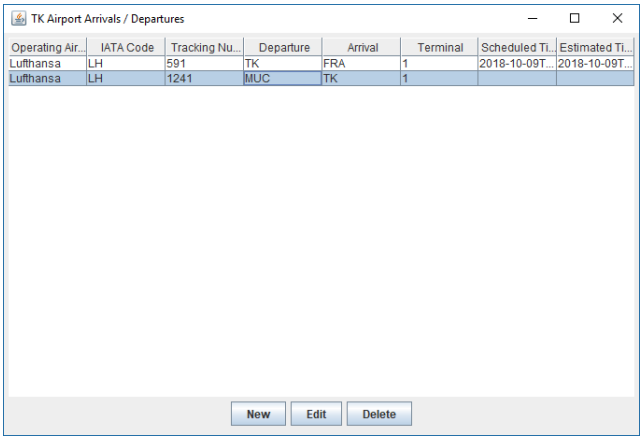
Interface Suggestions

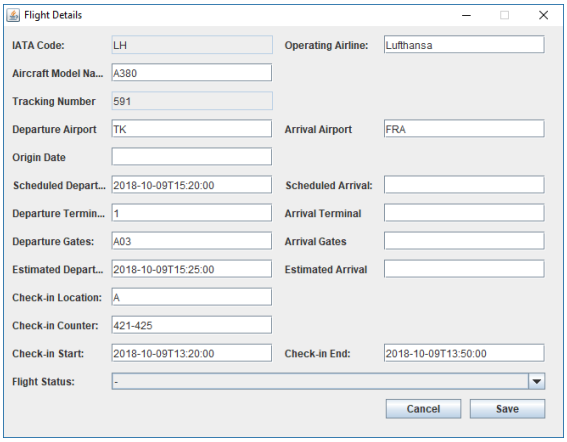
The following interfaces for the client and the server may (but not has to) look like this:

```
IFlightServer:
	void login(String clientName, IFlightClient client)
	void logout(String clientName)
	void updateFlight(String clientName, Flight flight)
	void deleteFlight(String clientName, Flight flight)
IFlightClient:
	void receiveListOfFlights(List<Flight> flights)
	void receiveUpdatedFlight(Flight flight, boolean deleted)
```

User Interface

May look like this:





Tutorials

- https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html
- https://www.tutorialspoint.com/java_rmi/index.htm
- https://www.eclipse.org/windowbuilder/