TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr. Max Mühlhäuser

TELEKOOPERATION
Fachbereich Informatik
Hochschulstr. 10
64289 Darmstadt

TK1: Distributed Systems - Programming & Algorithms

2nd Programming Exercise
Submission Date: 09.12.2020

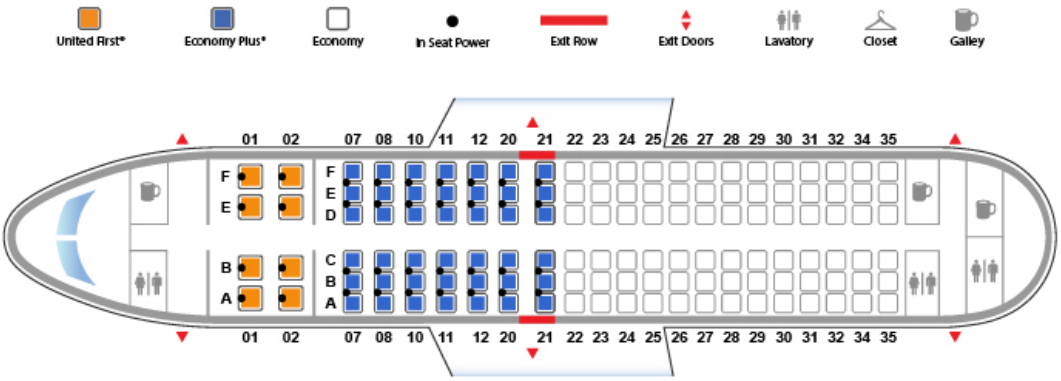
By handing in a solution you confirm that you are the exclusive author(s) of all the materials. Additional information can be found here: https://www.informatik.tu-darmstadt.de/en/students/studies/plagiarism/

P2. Plane Ticket Reservation using Web Services (20 P.)

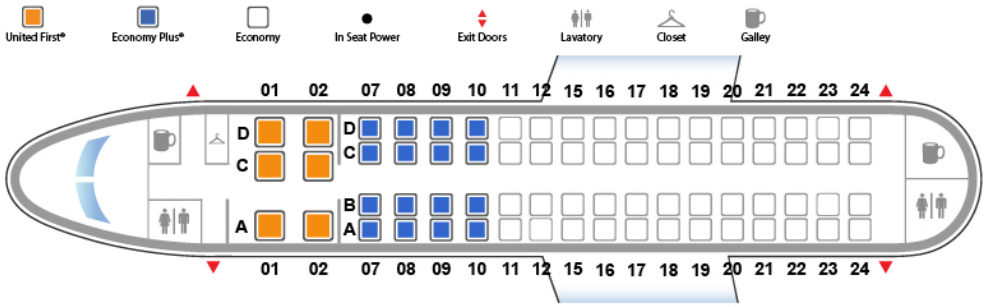
Again, consider the TK airport with three types of planes taking off three times every day.

A Boeing 737-900 with the following seat map,

an Airbus 319,



and Embraer E170.



There are 9 flights every day, 3 flights per type of plane taking off to different destinations. The plane ticket price is depending on two variables: the destination and the types of seat (e.g., First Class, Economy Plus Class, and Economy Class).

A customer can book one seat on one of the flights. After the selection of the flight (e.g., indicated by the flight number), the customer can select their seat. The price is determined based on the selection. *Emergency exit seats can only be selected by customers 40 years of age or less (this is an optional challenge).*

In addition to the seat the customer can select a preferred meal (Standard, Vegetarian, or Vegan).

Implement a web service for the plane ticket reservation at the airport.

The server application must provide the following functionality:

- The server must manage the reservation for every customer. Each reservation is stored using a unique ID for each customer. No complex session management is needed for the customer nor does the server need to persist the reservations on disk, i.e., no database needed.
- The server provides information about the upcoming flights and the available seats for the plane (row, seat number, type). Per day there are 9 flights, 3 per plane type. *Hint: As a simplification, consider only one week in your reservation system (this can be hardcoded). As a further simplification, the customer can only reserve a seat for one flight at a time.*
- The server must provide information about which seats are free for any flight at any given date.
- The server must only allow reservations when there are sufficient free seats remaining.

Each client application must provide the following functionality:

- The client application must provide a user interface, which shows the upcoming flights. After the customer selected a specific flight, the customer is shown a list (can be either graphical like depicted above or a textual list) of available seats and prices.
- The customer can reserve exactly one seat (row and seat number) for one flight. The customer can send his reservation using his “shopping cart”.
- The customer gets a return message, if a seat is not available, e.g., if another reservation might have been received beforehand.
- If the reservation is successful, the total price for the ticket is displayed based on the flight destination and the seat type. Payment does not need to be implemented; however a faked payment is nice.

Your task is to implement a SOAP web service which provides the described functionality. Also, implement a RESTful-Service, which provides the same functionality. Provide two clients, one for each web service.

Use JAX-WS and JAX-RS (and its reference implementation Jersey) as frameworks for the web services.

Hints:
Use the HTTP-Server for service deployment that is provided by the Java JDK (not included in the JRE).

Example for REST start up:

```
ResourceConfig rc = new ResourceConfig().packages("rest");
JdkHttpServerFactory.createHttpServer(URI.create("http://localhost:8080/"),
    rc);
```

Example for SOAP start up:

```
Endpoint.publish("http://localhost:8090/bookingservice", new
    ReservationBookingService());
```

Tutorials:

- http://openbook.galileocomputing.de/java7/1507_13_001.html#dodtp82d1ec9d-ccf4-456f-8af9-ebd4bb3c87b4 (German)
- http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example/ (English)
- http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/ (English)

Break down your solution into two separate projects:

- The "Clients" project contains both clients.
- The "Services" project contains the service implementations
    - The client project must not depend on the service source code (e.g., by importing its interfaces).
- Provide a Gradle script within the root folder of each project:
    - The default task in the "Services" project starts both the REST and the SOAP service.
    - The default task in the "Clients" projects starts one client for each service instance. In case of SOAP, the client script must download the WSDL file and generate the stubs first.
- Use the JDK tool wsimport for client stub generation.
- Start the REST server on port 8080. Start the SOAP server on port 8090.