

# CONCORDIA UNIVERSITY

## DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Fall 2016

Instructor: R. Jayakumar

### ASSIGNMENT 1

Issued: Sep. 26, 2016

Due: Oct. 10, 2016

---

**Note:** *The assignments must be done individually and submitted electronically.*

#### **Distributed Flight Reservation System (DFRS) using Java RMI**

In the assignments and project, you are going to implement a simple Distributed Flight Reservation System (DFRS): a distributed system used by passengers and managers to make and manage flight reservations between different cities. Consider three cities: Montreal, Washington and New Delhi. Flights can be booked between these cities by passengers and the number of flights available between these cities is limited and can be controlled by the managers. There are different classes of flights such as economy class, business class and first class and the number of seats for each class from one city to other is fixed.

When a passenger books a flight, the passenger's information is stored in a passenger record in the servers with following details: first name, last name, address, phone no., destination, class of flight, date of flight. These records are placed in several lists that are stored in a hash map according to the first letter of the last name indicated in the records. For example, all the records with the last name starting with "A" will belong to the same list and will be stored in a hash map (acting as the database) and the key will be "A". Each server also maintains a log containing the history of all the operations that have been performed on that server. This should be an external text file (one per server) and shall provide as much information as possible about what operations are performed, at what time and who performed the operation.

The users of the system are the passengers who want to book the flights and the managers who will add and edit the flights. Managers can be identified by a unique managerID, which is constructed from the acronym of the city and a 4-digit number (e.g. MTL1111). Whenever a manager performs an operation, the system must identify the city that manager belongs to by looking at the managerID prefix and perform the operation on that server. A manager should also maintain a log (text file) of the actions he/she performed on the system and the response from the system when available. For example, if you have 10 managers using your system, you should have a folder containing 10 logs.



Whenever a passenger wants to book the flight, he/she has to invoke the following operation on the specific city's server:

- *bookFlight (firstName, lastName, address, phone, destination, date, class) :*

When a passenger invokes this method on the server for his/her departure city through a client program, the server attempts to create a passenger record with the

information provided, assigns a unique *RecordID* and inserts the record at the appropriate position in the hash map. The server returns information to the manager whether the operation was successful or not and both the server and the client store this information in their logs.

Managers may invoke the following operations:

- *getBookedFlightCount* (*recordType*)


A manager invokes this method from his/her *ManagerClient* and the server associated with that manager concurrently finds out the number of records (i.e. number of flights booked) in all the cities/servers using UDP/IP sockets and returns the result to the manager. Note that only record counts (a number) are returned and not the records themselves. For example, if MTL has 6 records, Washington has 7 and New Delhi had 8, it should return the following: MTL 6, WST 7, NDL 8.

- *editFlightRecord* (*recordID*, *fieldName*, *newValue*)

When invoked by a manager, the server associated with this manager (determined by the unique *managerID*) can change the information about the flights available on a particular date from a specified city to a specified city. For example, the manager can change the information that flight going from Washington to New Delhi on September 24, 2016 at 1:00 p.m. has 55 economy seats, 20 business seats, and 15 fit class seats. He can even create a new flight at a specific date and delete the flight at a specific time.

Thus, this application has a number of *Servers* (one per city) each implementing the above operations for that city, passengers and managers invoking the operations at the associated *Server* as necessary. When a *Server* is started, it registers its address and related/necessary information with a central repository. For each operation, the *ManagerClient* finds the required information about the associated *Server* from the central repository and invokes the corresponding operation.

In this assignment, you are going to develop this application using Java RMI. Specifically, do the following:

- Write the Java RMI interface definition for the *Server* with the specified operations.
- Implement the *Server*. 
- Design and implement a *ManagerClient*, which invokes the server system to test the correct operation of the DFRS invoking multiple *Server* (each of the servers initially has a few records) and multiple managers.

You should design the *Server* maximizing concurrency. In other words, use proper synchronization that allows multiple passengers and managers to perform operations for the same or different records at the same time.

## Marking Scheme

[30%] *Design Documentation*: Describe the techniques you use and your architecture, including the data structures. Design proper and sufficient test scenarios and explain what you want to test. Describe the most important/difficult part in this assignment. You can use UML and text description, but limit the document to 10

pages. Submit the **documentation** and code by the due date; print the documentation and bring it to your demo.

**[70%]** *Demo in the Lab:* You have to register for a 5-minute demo. Please come to the lab session and choose your preferred demo time in advance. You cannot demo without registering, so if you did not register before the demo week, you will lose 40% of the marks. Your demo should focus on the following.

**[50%]** *Correctness of code:* **Demo your designed test scenarios to illustrate the correctness of your design.** If your test scenarios do not cover all possible issues, you'll lose part of mark up to 40%. You will also be evaluated on the implementation of your design.

**[20%]** *Questions:* You need to answer some simple questions (like what we've discussed during lab tutorials) during the demo. They can be theoretical related directly to your implementation of the assignment.

## Questions

If you are having difficulties understanding sections of this assignment, feel free to email the Teaching Assistant Mr. Harpreet Narula at harpreetnarula005@gmail.com. It is strongly recommended that you attend the tutorial sessions which will cover various aspects of the assignment.