CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Fall 2017 Instructor: R. Jayakumar

ASSIGNMENT 1

Issued: Sep. 21, 2017 Due: Oct. 5, 2017

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

Distributed Room Reservation System (DRRS) using Java RMI

In this assignment, you are going to implement a distributed room reservation system (DRRS) for a university with multiple campuses: a distributed system used by administrators who manage the availability information about the university's rooms and students who can reserve the rooms across the university's different campuses.

Consider three campus locations: Dorval-Campus (DVL), Kirkland-Campus (KKL) and Westmount-Campus (WST) for your implementation. The server for each campus must maintain a number of *RoomRecords*. A *RoomRecord* can be identified by a unique *RecordID* starting with RR (for *RoomRecord*) and ending with a 5 digits number (e.g. RR10000 for a *RoomRecord*).

A *RoomRecord* contains the following fields:

- Date.
- Room number.
- List of available time (applies every day).
- Booked_by (*studentID* if it's booked by student, null otherwise).
- You are allowed to add more if necessary.

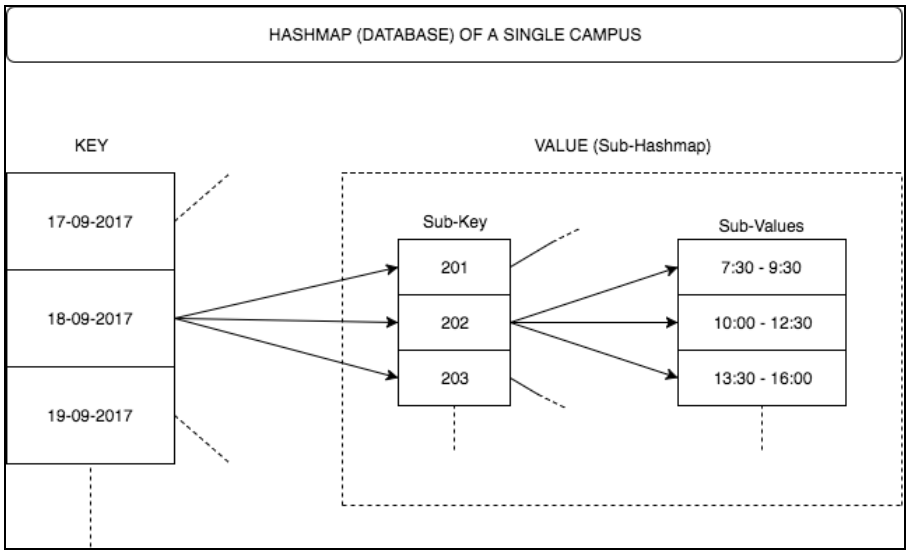

Figure 1 Hashmap of a single campus

The *RoomRecords* are maintained in a HashMap as shown in Figure 1. Here *Date* is the key, while the *value* is again a sub-HashMap. The key for sub-HashMap is the room number, while the value of the sub-HashMap is the list of time slots.

The users of the system are *administrators* and *students*. Administrators and students are identified by a unique *adminID and studentID* respectively, which is constructed from the acronym of the center and a 4-digit number (e.g. DVLA1111 for an admin and DVLS1111 for a student). Whenever the user performs an operation, the system must identify the server that user belongs to by looking at the *ID* prefix (i.e. DVLA or DVLS) and perform the operation on that server. The user should also maintain a log (text file) of the actions they performed on the system and the response from the system when available. For example, if you have 10 users using your system, you should have a folder containing 10 logs.

In this DRRS, there are different admins for 3 different servers. They create availability of rooms along with their time slots. A student can book a room at any campus, with any room number, on any date and time slot (if exists i.e. created and made available by an admin). A server (which receives the request) maintains a *booking-count* for every student. If this count reaches 3, then the student cannot book more rooms in that week. This counter resets every week. In other words, a student can make only three bookings per week. Moreover, if a room is booked, the student receives a unique *bookingId* (that is randomly generated by the server) as a confirmation. You should ensure that if a time slot is already booked by a student, then, another student cannot book the same time slot.

Each server also maintains a log file 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. These are some details that a single log file record must contain:

- Date and time the request was sent.
- Request type (create room, delete room, etc.).
- Request parameters (room number, time slots, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Admin Role:

The operations that can be performed by an admin are the following:

- *createRoom (room_Number, date, list_Of_Time_Slots):*

  When an admin invokes this method through a program called *AdminClient*, the server associated with this admin (determined by the unique *adminID* prefix) attempts to create a *RoomRecord* with the information passed, and inserts the record at the appropriate location in the hash map. The server returns information to the admin whether the operation was successful or not and both the server and the client store this information in their logs. If *room_Number* already exists in the database, it just adds the new time slots. If a time slot does not exist in the database, then add it. Log the information into the admin log file.

- *deleteRoom (room_Number, date, List_Of_Time_Slots)*

  When invoked by an admin, the server associated with this admin, (determined by the unique *adminID*) searches in the hash map to find and delete the room for the indicated date and time slots. Upon success or failure it returns a message to the admin and the logs are updated with this information. If a time slot does not exist, then obviously there is no deletion performed. Just in case that, if a room exists and is booked by the student, then, delete the room and reduce the student's *booking-count*. Log the information into the log file.

Student Role:

The operations that can be performed by a student are the following:

- *bookRoom (campusName, roomNumber, date, timeslot) :*

  When a student invokes this method from his/her campus through a client program called *StudentClient*, the server associated with this student (determined by the unique *studentID* prefix) attempts to create a *RoomRecord* with the information passed, assigns a unique *bookingID* and inserts the record at the appropriate location in the hash map. The server returns information to the student whether the operation was successful or not and both the server and the client store this information in their logs.

- *getAvailableTimeSlot (date):*

  A student invokes this method from his/her *StudentClient* and the server associated with that student concurrently finds out the number of records of available time slot on given date in the other servers using UDP/IP sockets and returns the result to the student. Please note that it only returns the record counts (a number) and not the records themselves. For example, if DVL has 6 records, KKL has 7 and WST had 8, it should return the following: DVL 6, KKL 7, WST 8.

- *cancelBooking (bookingID):*

  When invoked by a student, the server associated with this student, (determined by the unique *studentID*) searches the hash map to find the *BookingID* and cancels the booking. Upon success or failure it returns a message to the student and the logs are updated with this information. It is required to check that an occupied time slot can only be cancelled if it was booked by the same student who sends cancel request and has a valid *bookingID*.

Thus, this application has a number of *Servers* (one per campus) each implementing the above operations for that campus, *StudentClient* invoking the student's operations at the associated *Server* as necessary and *AdminClient* invoking the administrator's operations at the associated *Server*. When a *Server* is started, it registers its address and related/necessary information with a central repository. For each operation, the Student*Client/AdminClient* finds the required information about the associated *Server* from the central repository and invokes the corresponding operation. ***Your server should***

ensure that a student can only perform a student operation and cannot perform an admin 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 *StudentClient,* which invokes the server system to test the correct operation of the DRRS invoking multiple *Server* (each of the servers initially has a few records) and multiple students.
- Design and implement an *AdminClient,* which invokes the server system to test the correct operation of the DRRS invoking multiple *Server* (each of the servers initially has a few records) and multiple administrator.

You should design the *Server* maximizing concurrency. In other words, use proper synchronization that allows multiple users 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 electronically by the due date; print the documentation and bring it to your DEMO.

[70%] *DEMO in the Lab*: You have to register for a 5–10 minutes 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%] *The 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 will lose part of marks up to 40%.
- [20%] *Questions:* You need to answer some simple questions (like what we have 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 any aspect of this assignment, feel free to contact your teaching assistant (Mr. Parth Nayak at comp6231ta.dsd@gmail.com or Mr. Akash Shah at comp6231ta.as@gmail.com). It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered there.