CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Summer 2019 Instructor: Sukhjinder K. Narula

ASSIGNMENT 1

Issued: May 14, 2019 Due: Jun 3, 2019

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

Distributed Event Management System (DEMS) using Java RMI

In this assignment, you are going to implement a distributed event management system (DEMS) for a leading corporate event management company: a distributed system used by an event manager who manages the information about the events and customers who can book or cancel an event across the company's different branches.

Consider three branches in different cities: Toronto (TOR), Montreal (MTL) and Ottawa (OTW) for your implementation. The users of the system are *event managers* and *event booking customers*. Event Managers and customers are identified by a unique *managerID* and *customerID* respectively, which is constructed from the acronym of their branch's city and a 4 -digit number (e.g. TORM2345 for an event manager and TORC2345 for a customer) . Whenever the user performs an operation, the system must identify the server that the user belongs to by looking at the ID prefix (i.e.TORM or TORC) 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 DEMS, there are different event managers for 3 different servers. They create slots of events corresponding to event type as per the availability. There are three event types for which event slots can be created, Conferences, Trade shows and Seminars. A customer can book an event in any branch of the company, for any event type, if the event is available for booking (if the events slots corresponding to an event on a particular date is not yet full). A server (which receives the request) maintains a booking-count for every customer. A customer can not book more than one event with the same event id and same event type. There are three time slots available for each event type on a day- Morning(M), Afternoon(A) and Evening(E). An event id is a combination of city, time slot and event date (e.g. TORM100519 for a morning event on 10th May 2019 in Toronto, OTWA100519 for an afternoon event on 10th May 2019 in Ottawa and MTLE100519 for an evening event on 10th May 2019 in Montreal). You should ensure that if the availability of an event is full, more customers cannot book the event. Also, a customer can book as many events in his/her own city, but only at most 3 events from other cities overall in a month.

The *EventRecords* are maintained in a HashMap as shown in Figure 1. Here e*vent type* is the key, while the value is again a sub-HashMap. The key for sub-HashMap is the *eventID*, while the value of the sub-HashMap is the information about the event.

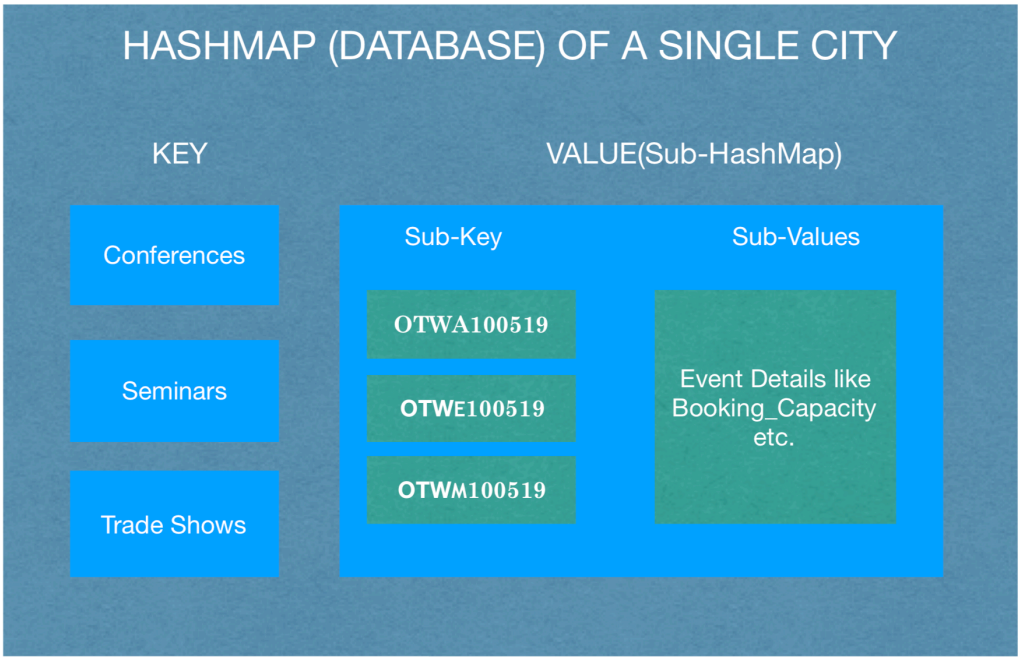

Fig. 1 Hashmap of a single city

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 (book an event, cancel an event, etc.).
- Request parameters (clientID, eventID, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Event Manager Role:

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

- *addEvent* (*eventID, eventType, bookingCapacity*):

  When an event manager invokes this method through the server associated with this event manager (determined by the unique *eventManagerID* prefix), attempts to add an event with the information passed, and inserts the record at the appropriate location in the hash map. The server returns information to the event manager whether the operation was successful or not and both the server and the client store this information in their logs. If an event already exists for same event type, the event manager can't add it again for the same event type but the new *bookingCapacity* is updated. If an event does not exist in the database for that event type, then add it. Log the information into the event manager log file.

- *removeEvent* (*eventID, eventType*):

  When invoked by an event manager, the server associated with that event manager (determined by the unique *eventManagerID*) searches in the hash map to find and delete the event for the indicated *eventType and eventID*. Upon success or failure it returns a message to the event manager and the logs are updated with this information. If an event does not exist, then obviously there is no deletion performed. Just in case that, if an event exists and a client has booked that event, then, delete the event and take the necessary actions. Log the information into the log file.
- *listEventAvailability* (*eventType*):

  When an event manager invokes this method from his/her branch's city through the associated server, that branch's city server concurrently finds out the number of spaces available for each event in all the servers, for only the given *eventType*. This requires inter server communication that will be done using UDP/IP sockets and result will be returned to the client. Eg: Seminars - MTLE130519 3, OTWA060519 6, TORM180519 0, MTLE190519 2.

Customer Role:

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

- *bookEvent* (*customerID, eventID, eventType*):

  When a customer invokes this method from his/her city through the server associated with this customer (determined by the unique *customerID* prefix) attempts to book the event for the customer and change the capacity left in that event. Also if the booking was successful or not, an appropriate message is displayed to the customer and both the server and the client stores this information in their logs.
- *getBookingSchedule* (*customerID*):

  When a customer invokes this method from his/her city's branch through the server associated with this customer, that city's branch server gets all the events booked by the customer and display them on the console. Here, bookings from all the cities, Ottawa, Montreal and Toronto, should be displayed.
- *cancelEvent (customerID, eventID):*

  When a customer invokes this method from his/her city's branch through the server associated with this customer (determined by the unique *customerID* prefix) searches the hash map to find the *eventID* and remove the event. Upon success or failure it returns a message to the customer and the logs are updated with this information. It is required to check that the event can only be removed if it was booked by the same customer who sends cancel request.

Thus, this application has a number of servers (one per city) each implementing the above operations for that branch, C*ustomerClient* invoking the customer's operations at the associated server as necessary and *EventManagerClient* invoking the event manager'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 *CustomerClient/EventManagerClient* finds the required information about the associated

server from the central repository and invokes the corresponding operation. ***Your server should ensure that a customer can only perform a customer operation and cannot perform any event manager operation, but an event manager can perform all operations.***

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 *CustomerClient,* which invokes the server system to test the correct operation of the DEMS invoking multiple servers (each of the servers initially have few records) and multiple customers.
- Design and implement a *EventManagerClient,* which invokes the server system to test the correct operation of the DEMS invoking multiple servers (each of the servers initially have few records) and multiple event managers.

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. The 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 (Ms. Harsh Deep Kour at harsh.comp6231@gmail.com or Mr. Kishan Bhimani at kishanbhimani9111@gmail.com or Mr. Iknoor Singh Arora at iknoor438@gmail.com). It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered there.