CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Winter 2019 Instructor: R. Jayakumar

ASSIGNMENT 1

Issued: Jan. 25, 2019 Due: Feb. 8, 2019

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

Distributed Library Management System (DLMS) using Java RMI

In this assignment, you are going to design and implement a Distributed Library Management System (DLMS) for a group of libraries: a distributed system used by library managers to manage the information about the items available in the libraries and library users to borrow or return items across the libraries.

Consider three libraries: Concordia (CON), McGill (MCG) and Montreal (MON) for your implementation. The users of the system are *library managers* and *library users* identified by a unique *managerID and userID* respectively, which is constructed from the acronym of their library and a 4-digit number (e.g. CONM1111 for a manager and CONU1111 for a user). Whenever the user performs an operation, the system must identify the server that user belongs to by looking at the *ID* prefix (i.e. CONM or CONU) 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 DLMS, there are different managers for the three libraries/servers. They create availability of items in their library along with the quantity of the items. A user can borrow an item offered by any library, if it is still available (if the item quantity is not yet zero). You should ensure that if the quantity of an item is zero, no more users cannot borrow that item. Also, a user can borrow multiple items in their own library, **but only 1 item from each of the other libraries.** Each item in a library has an associated *itemID* (a 4-digit number appended to the library acronym, like CON1012, MCG1023, MON1034, etc.) and an *itemName* (a string describing the item). Note that different libraries can have the item with the same *itemName* under different *itemID*s.

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 (borrow an item, return an item, etc.).
- Request parameters (*userID*, *itemID*, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

The information in a library are maintained in a hashmap as shown in Figure 1. All information must be kept only in the library servers, not in the clients.

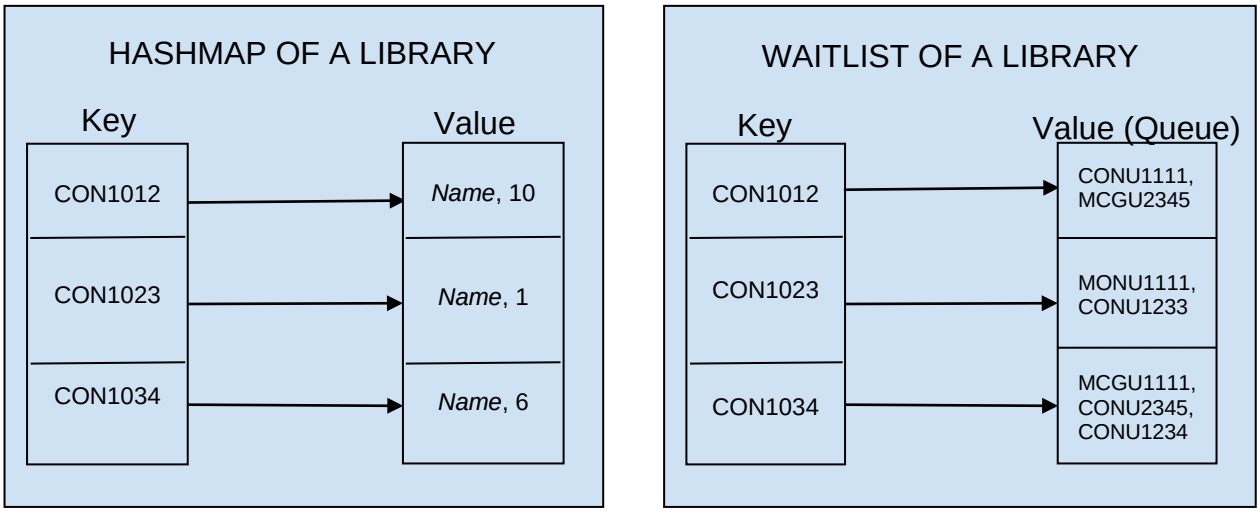

Fig. 1 Hashmap of a single library

Manager Role:

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

- *addItem (managerID, itemID, itemName, quantity):*

  When a manager invokes this method on the associated server (determined by the unique *managerID* prefix), it attempts to add an item with the information passed, and inserts the record at the appropriate location 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. If an item already exists, the new quantity entered should be added to the current quantity of the item. If an item does not exist in the database, then simply add it. Log the information into the manager log file.

- *removeItem (managerID, itemID, quantity)*

  When invoked by a manager, the server associated with this manager (determined by the unique *managerID*) searches in the hashmap to find and delete the item. There can be two cases of deletion, first, if the manager wants to decrease the quantity of that item, second, if the manager wants to completely remove the item from the library. Upon success or failure it returns a message to the manager and the logs are updated with this information. If an item does not exist, then obviously there is no deletion performed. Just in case that, if an item exists and a user has borrowed it, then, delete the item and take the necessary actions. Log the information into the log file.

- *listItemAvailability (managerID):*

  When a manager invokes this method from his/her library through the associated server, that library server finds out the names and quantities of each item available in the library. Eg: CON6231 Distributed Systems 5, CON6441 Advanced Programming 4, CON6491 Systems Software 0.

User Role:

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

- *borrowItem (userID, itemID, numberOfDays) :*

  When a user invokes this method from his/her library through the server associated with this user (determined by the unique *userId* prefix), it attempts to borrow the specified item. If the item is from a different library, then the user's library sends a UDP/IP request to the item's library to borrow. If the operation was successful, borrow the item and decrement the quantity for that item. Also, display an appropriate message to the user and both the server, the client stores this information in their logs. If the borrow operation is unsuccessful, ask the user if he/she wants to be added in the waiting queue. If prompted no, method ends, otherwise add the *userID* to the queue corresponding to the requested item and whenever the item is available again, automatically lend the item to the first user in that queue.

- *findItem (userID, itemName):*

  When a user invokes this method from his/her library through the server associated with this user, that library server gets all the *itemID*s with the specified *itemName* and the number of such items available in each of the libraries and display them on the console. This requires inter server communication that will be done using UDP/IP sockets and the result will be returned to the user. Eg: MON6231 5, CON6441 4, CON6497 1, MCG6132 5.

- *returnItem (userID, itemID):*

  When a user invokes this method from his/her library through the server associated with this user (determined by the unique *userID* prefix) searches the hash map to find the *itemID* and returns the item to its library. Upon success or failure it returns a message to the user and the logs are updated with this information. It is required to check that an item can only be returned if it was borrowed by the same user who sends the return request.

Thus, this application has a number of servers (one per library) each implementing the above operations for that library, *userClient* invoking the user's operations at the associated server as necessary and *managerClient* invoking the 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 *userClient*/*managerClient* finds the required information about the associated server from the central repository and invokes the corresponding operation. ***Your servers should ensure that a user can only perform a user operation and cannot perform a manager operation and vice-versa.***

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 *userClient,* which invokes the server system to test the correct operation of the DLMS invoking multiple servers (each of the servers initially has a few records) and multiple users.
- Design and implement a *managerClient,* which invokes the server system to test the correct operation of the DLMS 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 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%] ***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. Kritika Sharma at kritikasharma462@gmail.com or Mr. Pranav Bhatia at pb.comp6231@gmail.com or Mr. Kishan Bhimani at kishanbhimani9111@gmail.com). It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered there.