CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Fall 2018 Instructor: R. Jayakumar

ASSIGNMENT 1

Issued: Sep. 20, 2018 Due: Oct. 7, 2018

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

Distributed Course Registration System (DCRS) using Java RMI

In this assignment, you are going to implement a distributed course registration system (DCRS) for a university: a distributed system used by a program advisor who manages the information about the courses available in the university and students who can enroll or drop a course across the university's different departments.

Consider three departments: Computer Science (COMP), Software Engineering (SOEN) and Information Security (INSE) for your implementation. The users of the system are *program advisors* and *students*. Program advisors and students are identified by a unique *advisorID* and *studentID* respectively, which is constructed from the acronym of their department and a 4-digit number (e.g. COMPA1111 for an advisor and COMPS1111 for a student). 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.COMPA or COMPS) 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 DCRS, there are different advisors for 3 different servers. They create availability of courses along with the semester the course is available in. There are three semesters in a year, Fall, Winter and Summer. A student can enroll into a course offered by any department, in any semester, if it is still available (if the course capacity is not yet full). A server (which receives the request) maintains a *max-course count* for every student. If this count reaches 3, then the student cannot enroll into more courses in that semester. In other words, a student can enroll only up to 3 courses in a semester. You should ensure that if the capacity of a course is full, more students cannot book that course. Also, a student can take as many courses in his/her own department, but only at most 2 from other department courses overall.

The *CourseRecords* are maintained in a HashMap as shown in Figure 1. Here *Term* is the key, while the value is again a sub-HashMap. The key for sub-HashMap is the course, while the value of the sub-HashMap is the information about the course.

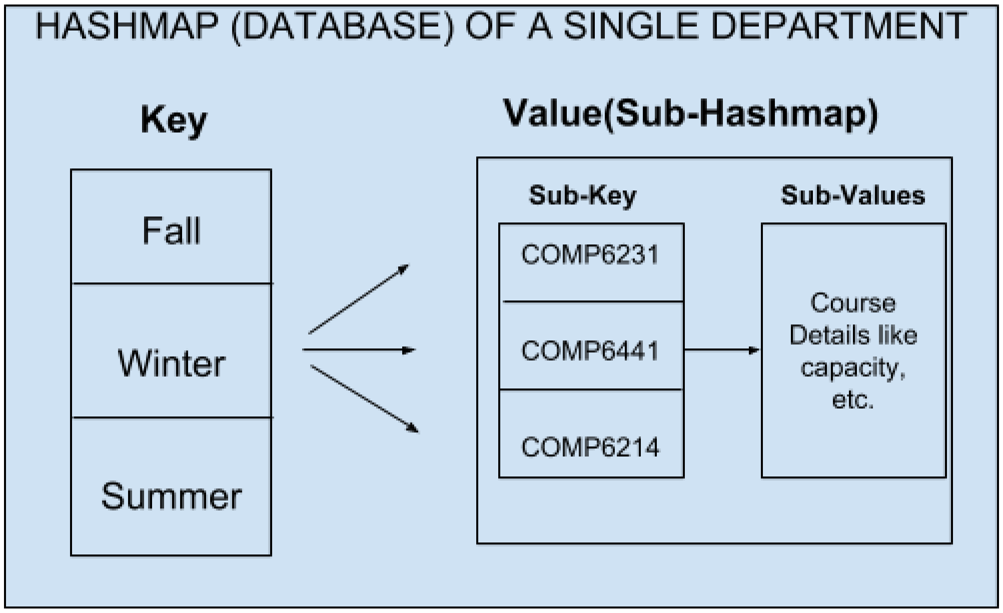

Fig. 1 Hashmap of a single department

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 (enrol in a course, drop a course, etc.).
- Request parameters (studentID, courseID, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Advisor Role:

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

- *addCourse* (*courseID, semester*):

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

- *removeCourse* (*courseID, semester*):

  When invoked by an advisor, the server associated with that advisor (determined by the unique *advisorID*) searches in the hashmap to find and delete the course for the indicated semester. Upon success or failure it returns a message to the advisor and the logs are updated with this information. If a course does not exist, then obviously there is no deletion performed. Just in case that, if a course exists and a student is enrolled into that course, then, delete the course and take the necessary actions. Log the information into the log file.

- *listCourseAvailability* (*semester*):

  When an advisor invokes this method from his/her department through the associated server, that department server concurrently finds out the number of spaces available for each course in all the servers, for only the given semester. This requires inter server communication that will be done using UDP/IP sockets and result will be returned to the student. Eg: Fall - COMP6231 5, SOEN6441 4, SOEN6497 0, INSE6132 5.

Student Role:

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

- *enrolCourse* (*studentID, courseID, semester*):

  When a student invokes this method from his/her department through the server associated with this student (determined by the unique *studentID* prefix) attempts to enroll the student in that course and change the capacity left in that course. Also if the enrolment was successful or not, an appropriate message is displayed to the student and both the server and the client stores this information in their logs.

- *getClassSchedule* (*studentID*):

  When a student invokes this method from his/her department through the server associated with this student, that department server gets all the courses enrolled by the student and display them on the console. Here, schedule from all the semesters, Fall, Winter and Summer, should be displayed.

- *dropCourse (studentID, courseID):*

  When a student invokes this method from his/her department through the server associated with this student (determined by the unique *studentID* prefix) searches the hash map to find the *courseID* and drops the course. 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 the course can only be dropped if it was enrolled by the same student who sends drop request.

Thus, this application has a number of servers (one per department) each implementing the above operations for that department, *StudentClient* invoking the student's operations at the associated server as necessary and *AdvisorClient* invoking the advisor'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 *StudentClient/AdvisorClient* 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 advisor operation, but an advisor 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 *StudentClient,* which invokes the server system to test the correct operation of the DCRS invoking multiple servers (each of the servers initially has a few records) and multiple students.
- Design and implement an *AdvisorClient,* which invokes the server system to test the correct operation of the DCRS invoking multiple servers (each of the servers initially has a few records) and multiple advisors.

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 (Ms. Kritika Sharma at sh.kritika01@gmail.com or Mr. Pranav Bhatia at pb.comp6231@gmail.com). It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered there.