CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

SOEN 423, Fall 2018 Instructor: R. Jayakumar

ASSIGNMENT 1

Issued: Sep. 18, 2018 Due: Oct. 2, 2018

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

Distributed Employee Management System (DEMS) using Java RMI

In the assignments and project, you are going to design and implement a simple Distributed Employee Management System (DEMS): a distributed system used by HR managers to manage information of Project Managers and Employees (Project Members) across different centers.

Consider three center locations: Canada (CA), United States of America (US) and United Kingdom (UK) for your implementation. The server for each center (called *CenterServer*) must maintain a number of Records. There are two types of Records: *ManagerRecord* and *EmployeeRecord*. A Record can be identified by a unique *RecordID* starting with MR (for ManagerRecord) or ER (for EmployeeRecord) and ending with a 5 digits number (e.g. MR10000 for a ManagerRecord or ER10001 for an EmployeeRecord).

The ManagerRecord contains the following fields:

- First Name
- Last Name
- Employee ID
- Mail ID
- Project Information
  - Project ID (e.g. P00001)
  - Name of the Client
  - Name of the Project
- Location (CA, US, UK)

The EmployeeRecord contains the following fields:

- First Name
- Last Name
- Employee ID
- Mail ID
- Project ID

The 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 an “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” as value as record. We do not distinguish between ManagerRecords and EmployeeRecords when inserting them into the hash map (i.e. a list may contain both manager and employee records). 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 timestamp of operation and who performed the operation.

The users of the system are HR managers. They can be identified by a unique *ManagerID*, which is constructed from the acronym of the center and a 4-digit number (e.g. CA1111). Whenever a manager performs an operation, the system must identify the center that manager belongs to by looking at the ManagerID prefix and perform the operation on that server. The managers carry with them 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 managers using your system, you should have a folder containing 10 logs.

The operations that can be performed are the following:

- *createMRecord* (*firstName, lastName, employeeID, mailID, projects*):

  When a manager invokes this method from his/her center through a client program called ManagerClient, the server associated with this manager (determined by the unique ManagerID prefix) attempts to create a ManagerRecord with the information passed, assigns a unique RecordID 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.

- *createERecord* (*firstName, lastName, employeeID, mailID, projectID, location*):

  When a manager invokes this method from a ManagerClient, the server associated with this manager (determined by the unique ManagerID prefix) attempts to create an EmployeeRecord with the information passed, assigns a unique RecordID 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.

- *getRecordCounts*():

  A manager invokes this method from his/her ManagerClient and the server associated with that manager concurrently finds out the number of records (both MR and ER) in the other centers using UDP/IP sockets and returns the result to the manager. Please note that it only returns the record counts (a number) and not the records themselves. For example, if CA has 6 records, US has 7 and UK had 8, it should return the following: CA 6, US 7, UK 8.

- *editRecord* (*recordID, fieldName, newValue*):

  When invoked by a manager, the server associated with this manager, (determined by the unique managerID) searches in the hash map to find the recordID and change the value of the field identified by "fieldname" to the newValue, if it is found. Upon success or failure, it returns a message to the manager and the logs are updated with this information. If the new value of the fields such as mailID is not specified, it is invalid. For example, if the found Record is a ManagerRecord and the field to change is location and newValue is other than CA, US or UK, the server shall return an error. The fields that should be allowed to change are mailID, project information and location (for ManagerRecord), and mailID, projectID(for EmployeeRecord).

Thus, this application has a number of CenterServers (one per center) each implementing the above operations for that center and ManagerClients (one per center) invoking the manager's operations at the associated CenterServer as necessary. When a CenterServer 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 CenterServer 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 CenterServer with the specified operations.
- Implement the CenterServer.
- Design and implement a ManagerClient, which invokes the center's server system to test the correct operation of the DEMS invoking multiple CenterServer (each of the servers initially has a few records) and multiple managers.

You should design the CenterServer maximizing concurrency. In other words, use proper synchronization that allows multiple HR manager 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 enough 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.

[70%] ***DEMO in the Lab***: You have to register for the 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 (similar to those discussed during tutorials) during the demo. They can be theoretical related directly to your implementation of the assignment.

Submission:

Submit the document and code electronically on the ENCS Electronic Assignment Submission (EAS) system by midnight on the due date; print the documentation and bring it to your DEMO.

Questions:

If you are having difficulties understanding sections of this assignment, feel free to email/contact your teaching assistant Mr. Shivaraj Alagond at shivaraj.alagond@gmail.com. It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered.