CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Winter 2014 Instructor: R. Jayakumar

ASSIGNMENT 1

Issued: Jan. 21, 2014 Due: Feb. 4, 2014

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

Distributed Player Status System (DPSS) using Java RMI

In the assignments and project, you are going to implement a simple Distributed Player Status System (DPSS): a distributed system to manage player status across multiple game servers used by players and administrators.

Consider three geo-locations: North-America (NA), Europe (EU) and Asia (AS) for your implementation. The server for each geo-location (called *GameServer*) must maintain a number of player accounts. An *Account* contains specific information (first name, last name, age, password of at least 6 characters and IP-address) about a *Player* and is identified by a unique username. A *Username* has a minimum length of 6 characters and a maximum length of 15 characters. The accounts are placed in several lists that are stored in a hash table according to the first character of the username indicated in the account. For example, all the accounts with the username starting with an "A" will belong to the same list and will be stored in a hash table (acting as the database) using the key "A". 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 what time and who performed the operation.

The system has two distinct types of users: *Player*s and *Administrator*s. *Player*s can be identified by their username, password and IP-address. Whenever a player performs an operation, the system must identify the geo-location that the player belongs to based on his/her IP-address and perform the operation on that server. The players 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 players using your system, you should have a folder containing 10 logs. Administrators can be identified by their username, password and IP-address. For the sake of this assignment all the administrators have the same username which is *Admin* and the password is also *Admin*. The administrators also maintain a log.

IP-Addresses

For this assignment, there are only three possible ranges of IP-addresses:

1. 132.xxx.xxx.xxx : IP-addresses starting with 132 indicate a North-American geo-location.
2. 93.xxx.xxx.xxx : IP-addresses starting with 93 indicate an European geo-location.
3. 182.xxx.xxx.xxx : IP-addresses starting with 182 indicate an Asian geo-location.

The operations that can be performed are the following:

Operations performed by Players

- *createPlayerAccount* (*FirstName, LastName, Age, Username, Password, IPAddress*)

  When a player invokes this method from his/her geo-location through a client program called *PlayerClient*, the server associated with this player (determined by the *IPAddress*) attempts to create an account with the information passed if the username does not exist and that the passed information is valid according to the problem description, and inserts the account at the appropriate location in the hash table. The server returns information to the player whether the operation was successful or not and both the server and the player store this information in their logs.

- *playerSignIn* (*Username, Password, IPAddress*)

  When a player invokes this method from his/her geo-location through a client program called *PlayerClient*, the server associated with this player (determined by the *IPAddress*) attempts to verify if the account exists, that the password matches the account password and that the account is not currently signed-in. If these conditions are met, the server sets the account status to online and returns a confirmation to the player. Otherwise a descriptive error is returned. Both the server and the player store this information in their logs. (There are many ways this can be done. You are encouraged to design your system in such a way that it is simple to distinguish between online and offline accounts without impacting the performance of the system.)

- *playerSignOut* (*Username, IPAddress*)

  When a player invokes this method from his/her geo-location through a client program called *PlayerClient*, the server associated with this player (determined by the *IPAddress*) attempts to verify if the account exists and that the account is currently signed-in. If these conditions are met, the server sets the account status to offline and returns a confirmation to the player. Otherwise a descriptive error is returned. Both the server and the player store this information in their logs.

Operation performed by Administrators

- *getPlayerStatus* (*AdminUsername, AdminPassword, IPAddress*)

  When an administrator invokes this method from his/her geo-location through a client program called *AdministratorClient*, the server associated with this administrator (determined by the *IPAddress*) attempts to, if the credentials are valid, concurrently find out the number of online players and offline players in the other geo-locations using UDP/IP sockets and returns the result to the administrator. Please note that it only returns the player counts (a number) and not the player information. For example, if NA has 6 players online, EU has 7 players online and AS has 8 players online and they each have 1 player offline it should return the following (which should also be stored in the server and administrator log): NA: 6 online, 1 offline. EU: 7 online, 1 offline, AS: 8 online, 1 offline.

Thus, this application has a number of *GameServers* (one per geo-location) each implementing the above operations for that geo-location, *PlayersClient* (one per player) invoking the player's operations at the associated *GameServer* as necessary and *AdministratorsClient* (one per administrator) invoking the administrator's operations at the associated *GameServer*. When a *GameServer* is started, it registers its address and related/necessary information with a central repository. For each operation, the *PlayersClient/AdministratorClient* finds the required information about the associated *GameServer* 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 *GameServer* with the specified operations.
- Implement the *GameServer*
- Design and implement a *PlayerClient* which invokes the geo-location's server system to test the correct operation of the DPSS invoking multiple *GameServer* (each of the servers initially has a few accounts) and multiple players.
- Design and implement an *AdministratorClient* which invokes the geo-location's server system to test the correct operation of the DPSS invoking multiple *GameServer* (each of the servers initially has a few accounts) and multiple administrators.

You should design the *GameServer* maximizing concurrency. In other words, use proper synchronization that allows multiple users to perform operations for the same or different accounts 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 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-minute 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%] C*orrectness 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'll lose part of mark up to 40%. You will also be evaluated on the implementation of your design.

[20%] *Questions:* You need to answer some simple questions (like what we've 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 sections of this assignment, feel free to email the Teaching Assistant at alexandre.hudon@sympatico.ca. It is strongly recommended that you attend the tutorial sessions which will cover various aspects of the assignment.