Assigned: Sunday April 20, 2014 **Due**: on or before 9:00 PM on **Friday May 9, 2014**

When doing this homework, first create a directory named `HW5` somewhere in your home directory (e.g. as a subdirectory of a `cse305` directory). Place your solution to each question in a file or set of files, as indicated, in the `HW5` directory. When you're ready to submit, zip up the `HW5` directory *and its contents* (use the –r flag for the zip command – see 'man zip'), and use the `submit_cse305` command to submit your `HW5.zip` file.

It is very important that you pay close attention to the naming conventions for files and directories for your homework submissions in this course. Having uniform names for all student submissions makes grading submissions much easier. If you do not adhere to the naming conventions, grading of your work will be delayed, or it may simply be returned to you ungraded for you to correct the names. Or, you may get zero.

Distributed programming in Erlang and Java

This homework assignment has a "standard" part and an "extended" part. Completing the standard part can earn you full points. Completing the extended part can earn you extra credit. Note the point assignments. Note also how grading will be done (i.e. with a grading demo).

Standard requirements (50 points - 30 points for the first language, 20 points for the second language)

Write a distributed chat system consisting of one server and an arbitrary number of clients. Chats are between two clients only. The server keeps track of clients who are on-line; clients go on-line by registering with the server. Clients must register with unique usernames when they go on-line. If the username they wish to register under is already in use (by an existing on-line client) the request to go on-line fails.

Sample commands:
client> goOnline(Server, Client, Username).
client> goOffline(Server, Client, Username).

When an on-line client (the originating client) wishes to chat with another client (the target client), it requests a chat connection from the server. The request is to chat with a given username. If that username is not associated with an on-line client the originating client is told there is no such user on-line at the moment. If that username is associated with an on-line client the server asks that client if they wish to chat with the originating client. If the answer is no then no connection is made, and the originating client is told that the connection request is refused. If the answer is yes the originating client is given the target client's address/PID and from that point the two clients communicate directly with one another, without the server's intervention.

Sample commands:
client> requestChatWith(Server, OriginatingClient, TargetClientUsername).
client> acceptChatRequest(Server,OriginatingClientUsername,TargetClientUsername).
client> rejectChatRequest(Server,OriginatingClientUsername,TargetClientUsername).

A two-client chat continues until one of the two quits the chat. At that point the two-way connection is severed.

Sample commands:
client> sendMessage("Let's go to Duff's for wings tonight!").
client> quitChat().

Extended requirements (25 points – 15 points for first language, 10 points for second language)

There can be several servers and several clients. Chats can involve multiple clients.

Some servers, running at known IP addresses, are so-called root servers. Additional servers can be set up. Each server knows all other servers that are up and running. When a new server starts up it registers with one of the root servers, which when ensures that all servers have a connection to this new server (and vice-versa).

Each works as described above – except that not every server has complete information about all clients. When a client wishes to go on-line with a given username the server asks all other servers whether the name is in use. If the server receives negative responses from all other servers it registers the client with that username; otherwise that name is rejected.

When a client requests to chat with another client registered on the same server the connection proceeds as above. If the other client is not registered on the current server the server broadcasts the request to all other servers. If a positive response comes back the connection is established between clients. If no response is received within some timeout threshold (adjustable at root server start-up --- all servers abide by the same timeout) then no connection is established.

If a client requests a connection with a client already involved in a chat and that client accepts the chat request the new client is added to the chat – every client involved in the chat knows all other clients in the chat. When a client leaves a chat the chat continues between the other participants, unless there is just one participant left in the chat.

Port numbers

In Java you will need to specify port numbers for communication. To ensure you don't use someone else's, use the port number(s) given by this formula:

(50000 + SPNO) + 128 * *n*

where SPN is the Starting Port Number Offset (given in the Port Number column in UBLearns), and *n* is the number of the port that you wish to use (start counting at zero). For example, if your Starting Port Number Offset is 103, and you will need 3 ports, use these three:

(50000+103) + 128*0 = 50103
(50000+103) + 128*1 = 50231
(50000+103) + 128*2 = 50359

Partners

You may (but need not) work with a partner. You are strongly encouraged to work with a partner. Each member of a pair will earn the same grade. A pair has as many late days as the partner with the fewest late days remaining.

You must document in your code who the partners were, and who is the primary author on each function/class/etc.

Grading demo

You must make an appointment with Bing to demo your chat application for grading purposes. Bing will specify on which hosts you will start your server(s) and client(s). You will then demonstrate the functionality of your system. Your demo must last no longer than 10 minutes. If you run into technical difficulties during your demo you may be asked to reschedule. You are encouraged to demo your code before the submission deadline – in which case you will submit the code your demoed. If you schedule a demo after the deadline you will demo the code you submitted.