SEG2105. Introduction to Software Engineering

Laboratory 3 – OCSF and SimpleChat (1%)

Fall 2016

Lab date: Week of September 26th.

Exercise due: **The TA will verify your work at the end of your lab. No submissions are required.**

Instructions for setting up the OCSF and SimpleChat code in Eclipse are found here:

http://www.site.uottawa.ca/~mgarz042/SEG2105/assignments/Lab2_Getting_Started.html

Group work and partners: For this lab, you may have a different partner from the one you had for the first lab, but please only do so after discussing with your TA and your previous partner, so we don't end up with partners that are stranded. In other words, do not abandon your partner without ensuring your partner has someone else to work with. You must work with the same partner from this point onwards in the course, unless your partner drops the course.

Purpose of the lab: These lab questions are designed to help you become very familiar with the design of the *SimpleChat* program and the *OCSF* framework. You should read the relevant sections of the textbook before coming to the lab. You will be tested on your understanding of SimpleChat and OCSF in the midterm.

Testcase descriptions are available online:

http://www.site.uottawa.ca/school/research/lloseng/supportMaterial/source/code/simplechat1/Testcases-Phase1.html

Exercises to do: Work on the exercises below. The following exercises will help you to become familiar with the internals of OCSF and Phase 1 of an instant messaging application we call SimpleChat. Modify the application to provide the following features (**Remember: do not modify the OCSF framework**).

Prerequisite: On the book's web site, you will find a set of 'test cases' for Phase 1 of the SimpleChat program. We will discuss test cases in much more detail in Chapter 10. For now, you can simply see them as a set of instructions that allow you to verify the functionality of the system. You can also use them to learn about the system. Pick **ten** Phase 1 test cases and execute them.

Client side:

a) **Currently, if the server shuts down while a client is connected, the client does not respond**, and continues to wait for messages. Modify the client so that it responds to the shutdown of the server by printing a message saying the server has shut down, and quitting. Design hint: look at the methods called *connectionClosed* and *connectionException*.

b) **The client currently always uses a default port**. Modify the client so that it obtains the port number from the command line. Design hint: look at the way it obtains the host name from the command line. Test that this works by connecting a client to a server using a different port from the default. If the port is omitted from the command line, then the default value should still be used.

Server side:

c) **Currently the server ignores situations where clients connect or disconnect**. Modify the server so that it prints out a nice message whenever a client connects or disconnects. Hint: you will simply have to write code in *EchoServer* that overrides certain methods found in *AbstractServer* – study the *AbstractServer* description above to determine which methods you have to override.

What to show to your TA:

- Results of execution of test cases.
- Changes for client and server side. Both team members must know about the changes made to the code.