THERE IS A ZERO TOLERANCE CHEATING POLICY

ANY HONOR CODE VIOLATION – NO MATTER HOW SLIGHT – WILL RESULT IN AN F IN THE COURSE AND REFERRAL TO THE OFFICE OF STUDENT CONDUCT

CSE 5306 **Lab # 2** **Summer 2020**

Objectives:

1. Exposure to asynchronous messaging.
2. Implementing basic access transparency (converting units of length).

Due: Sunday, August 9th, before 11:59pm via Canvas

Project Specification:

These will be **individual** projects. You may write the program in any language that is supported under any Integrated Development Environment (IDE). Keep in mind that available controls, objects, libraries, et cetera, may make some of these tasks easier in one language than in another. Finally, because of the lack of restrictions on IDEs, you will have to have that IDE available to demo to the TA (e.g., you will demo the program on your own laptop).

All components should be managed with a *simple* GUI. The GUI should provide a way to kill the process without using the 'exit' button on the window.

Terminology used in this lab is described in **§4.3.2 'Message-oriented Persistent Communications'** of Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms, 2 ed.

You will write an asynchronous messaging system consisting of a message broker (server) and three client processes. Each client process will connect to the server over a socket connection. The server should be able to handle all three clients simultaneously and display the status of the connected clients in real time. How to distinguish clients is left to the developer's discretion.

Upon startup, clients will prompt users to select one of two options:

1. Upload message; or,
2. Check for messages.

If a user selects 'Upload message', the client will prompt the user to input a decimal number, which will represent a length in meters. The client will prompt the user to designate an output queue (described below), then proceed to upload the message to the server.

If a user selects 'Check for messages', the client will prompt the user to select an output queue. The client will proceed to poll the output queue. If no messages are present in the queue, the user should be notified that the queue is empty. If any messages are available, those messages will be retrieved from the queue and displayed to the user.

The message broker will maintain three output queues: A, B, and C. Upon placing a message into the queue, the server will perform a conversion of the input length received from the client into several different units of length. The corresponding units of length for each queue are as follows.

A	B	C
• Meter	• Parsec	• Mile
• Millimeter	• Light Year	• Nautical Mile
• Centimeter	• Inch	• American football field
• Kilometer	• Foot	• Hand
• Astronomical Unit	• Yard	• Horse

Yes – 'hand' and 'horse' are actual units of length. Look them up.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

Each client may upload messages to, or retrieve messages from, any of the three queues. The message broker will maintain a repository of conversion rules to translate between the aforementioned units of length. The repository should be maintained in a simple text file.

When the message broker receives a message from a client, it will check the designated output queue. The broker will then convert the value received from the client into each of the metrics corresponding to that queue. Messages placed in the output queue will consist of the input length converted into each of the metrics assigned to that queue.

Contents of the queue should be persistent (e.g., they should survive a process restarting). Messages retrieved from a queue should be deleted from the server (e.g., removed from both volatile and non-volatile memory).

Values in the repository should be editable using a text editor provided by the operating system (i.e., Notepad in Windows or vi in Linux). Contents of the repository will be read into the message broker upon startup. Values will not be changed during runtime. Metrics will not be added, deleted, or switched to a different queue, but may be rearranged in the repository in an arbitrary order.

The required actions are summarized below.

Client

Startup:

1. Prompt the user to 'Upload Message' or 'Check for Messages'.
2. Connect to the server over a socket and proceed to carry out the user's request.
3. Return to **Client Startup: Step 1** until manually killed by the user.

Upload Messages:

1. Prompt the user to input a length in meters.
2. Prompt the user to select an output queue.
3. Connect to the server and notify the user of the active connection.
4. Upload the message to the server and notify user of message submission.
5. Return to **Client Startup: Step 1**.

Check for Messages:

1. Prompt the user to select a message queue to check.
2. Connect to the server and poll the queue.
    a. If the queue is empty, notify the user.
    b. If the queue is not empty, retrieve all messages and display contents to user.
3. Return to **Client Startup: Step 1**.

Server

The server should support three concurrently connected clients. The server should indicate which clients are presently connected on its GUI. How to distinguish clients is left to the developer's discretion.

The server will execute the following sequence of steps:

1. Startup and listen for incoming connections.
2. Print that a client has connected.
    a. If a client has connected to upload messages:
        i. Print the input value and designated output queue to the GUI.

   ii. Perform the length conversions according to the designated output queue and print the conversions to the GUI.
   iii. Place the message in the output queue.
b. If a client has connected to check for messages:
   i. Print which output queue the client is polling.
   ii. If messages are retrieved by the client, delete those messages from the queue.
3. Return to **Step 1** until manually killed by the user.

Notes:

- All three clients and the server may run on the same machine.
- The server must correctly handle an unexpected client disconnection without crashing.
- When a client disconnects from the server, the server GUI must indicate this to the user in real time.
- The program must operate independently of a browser.

Citations:

You may use open source code found on the Internet in your program. When citing this code:

YOU MUST CITE THE EXACT URL TO THE CODE IN THE METHOD / FUNCTION / SUBROUTINE HEADER WHERE THE CODE IS UTILIZED.

Failure to cite the exact URL will result in a twenty (20) point deduction on the grade of your lab

A full list of your source URLs should be included in your writeup file. Including generic citations (for instance, simple listing "StackOverlow.com" without additional details) will result in a ten (10) point deduction in your lab grade, per instance.

Submission Guidelines:

FAILURE TO FOLLOW ANY OF THESE DIRECTIONS WILL RESULT IN DEDUCTION OF SCORES

Submit your assignment via the submission link on Canvas. You should zip your source files and other necessary items like project definitions, classes, special controls, DLLs, et cetera and your writeup into a single zip file. No other format other than zip will be accepted. The name of this file should be **lab#_lastname_loginID.zip**. Example: If your name is John Doe and your login ID is jxd1234, your submission file name must be "lab#_doe_jxd1234.zip" where # is the number of the lab assignment.

Be sure that you include everything necessary to unzip this file on another machine and compile and run it. This might include forms, modules, classes, configuration files, et cetera. DO NOT INCLUDE ANY RUNNABLE EXECUTABLE (binary) program. The first two lines of any file you submit must contain your name and student ID.

You may resubmit the project at any time. Late submissions will be accepted at a penalty of 10 points per day. If your program is not working by the deadline, send it anyway and review it with the TA for partial credit. Do not take a zero or excessive late penalties just because it isn't working yet. We will make an effort to grade you on the work you have done.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

CSE 5306 **Lab # 2** **Summer 2020**

Writeup:

Your write-up should include instructions on how to compile and run your program. Ideally it should be complete enough that the TA can test your program without your being there. Your writeup should include any known bugs and limitations in your programs. If you made any assumptions, you should document what you decided and why. This writeup can be in a **docx** or **pdf** format and should be submitted along with your code.

Grading:

Points – element:

15 – Client process works correctly.
15 – Server process works correctly.
05 – Server indicates which clients are presently connected in real time.
15 – Server correctly handles message persistence.
15 – Metrics in server repository may be arbitrarily reordered.
10 – Server correctly translates between units of measure.
15 – Client correctly displays output messages.
05 – Client and server handle disconnections correctly.
05 – Comments in code.

Deductions for failing to follow directions:

-10 – Late submission per day.
-05 – Including absolute/ binary/ executable module in submission.
-02 – Submitted file doesn't have student name and student ID in the first two lines.
-05 – Submitted file has a name other than student's lastname_loginID.zip.
-05 – Submission is not in zip format.
-05 – Submitting a complete installation of the Java Virtual Machine.
-10 – Per instance of superfluous citation.
-30 – Server and/or clients do not have their own GUI.

To receive full credit for comments in the code you should have headers at the start of every module / function / subroutine explaining the inputs, outputs, and purpose of the module. You should have a comment on every data item explaining what it is about. (Almost) every line of code should have a comment explaining what is going on. A comment such as /* Add 1 to counter */ will not be sufficient; the comment should explain what is being counted.

Important Note:

You may discuss the problem definition and tools with other students. You may discuss the lab requirements. You may discuss or share project designs. All coding work must be your own. You may use any book or web reference as long as you cite that reference in the comments. If we detect that portions of your program match portions of any other student's program, it will be presumed that you have colluded unless you both cite a specific source for the code.

You must not violate University of Texas at Arlington regulations, laws of the State of Texas or the United States, or professional ethics. Any violations, however small, will not be tolerated.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

DO NOT POST YOUR CODE ON PUBLICLY ACCESSIBLE SECTIONS OF WEBSITES UNTIL AFTER THE COURSE CONCLUDES. SHOULD YOU DO SO, THIS WILL BE CONSIDERED COLLUSION, AND YOU WILL BE REFERRED TO THE OFFICE OF STUDENT CONDUCT AND RECEIVE A FAILING GRADE IN THE COURSE