1. [10] Create a new Eclipse project named ChatServerAuthenticated. Create a ChatServerAuthenticated class for the project with all of the capabilities of your ChatServer class from Homework 5.

2. [10] Create a class named SocketConnection which will maintain a Socket instance, the user’s name, and a state variable which indicates that the SocketConnection instance is in one of three states: “awaiting password”, “awaiting username”, or “ready”. When created, a SocketConnection instance should be in the “awaiting password” state and write “Enter server password:” to the Socket. The signature of your SocketConnection constructor should be “public SocketConnection(Socket socket) throws IOException”. All instance variables should be private within the SocketConnection class and your SocketConnection class should not include any methods not specified by the assignment. Your ChatServerAuthenticated class should maintain a Vector of SocketConnection instances instead of a Vector of Sockets as with your original ChatServer class.

3. [5] Add a “public boolean isClosed()” method to your SocketConnection class which returns true if the Socket is closed.

4. [5] Add a “public void close() throws IOException” method to your SocketConnection class which closes the Socket.

5. [30] Add a “public String read() throws IOException” method to your SocketConnection class which reads available input from the Socket, if available, and returns the String that should be written to all SocketConnection instances, if any such output exists.

If the SocketConnection instance is in the “awaiting password” state and the input is the server password (which can be anything you’d like; I used “7777”), send the message “Correct password, enter name:” to the Socket, change the state to “awaiting username”, and return null. If the password was incorrect, send the message “Incorrect password, try again:” to the Socket, don’t change the state, and return null.

If the SocketConnection instance is in the “awaiting username” state, set the SocketConnection’s username variable to the input, change the state to “ready”, and return the user name plus “ has joined.”, indicating that this text should be sent to all SocketConnection instances.

If the SocketConnection instance is in the “ready” state, return the username plus “> “ plus the input, indicating that this text should be sent to all SocketConnection instances.
Here’s a screenshot of this interaction:

![Chat Client](image)

You’ll want to disable the “... Joined.” message sent by your client classes because that message will be interpreted as a password attempt.

6. [10] Add a “public void write(String s) throws IOException” method to your SocketConnection class which writes the String s to the Socket only if the SocketConnection instance is in the “ready” state and does nothing otherwise.


9. [+20] Modify the behavior of your SocketConnection’s read() method such that it will close the Socket after three incorrect password attempts. This should result in the SocketConnection instance being removed from the ChatServerAuthenticated’s Vector of current SocketConnection instances.

Zip your authenticated server project and problem-set solution document (PDF or plain text format). Name your zip file <your full name>Homework7.zip and upload to Blackboard.