The goal of this assignment is to provide experience working with state machines in a realistic application and with refactoring software. The book exercises are intended to provide practice working with Turing machines.


3. [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.

4. [10] Create an inner class within your ChatServerAuthenticated 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 username”, “awaiting password”, or “ready”. When created, a SocketConnection instance should be in the “awaiting username” state and write “Enter username:” 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.

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

6. [5] Add a “public void close() throws IOException” method to your SocketConnection class which closes the Socket.

7. [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 username” state, set the SocketConnection’s username variable to the input, change the state to “awaiting password”, send the message “Enter password:” to the Socket, and return null (i.e., no message to be sent to all connected Sockets).
If the SocketConnection instance is in the “awaiting password” state, use a PasswordList instance (i.e., from previous assignments) to verify that the input password matches the username. If the password is correct, change the state to “ready” and return the user name plus “ has joined.” (i.e., so that this can be sent to all connected Sockets). If the password was incorrect, send the message “Incorrect password, try again:” to the Socket, don’t change the state, and return null. If there is no password entry for the username in the PasswordList, the server should simply never accept a password.

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 connected Sockets.

Here’s a screenshot of this interaction:

![Screenshot](image)

Note that you’ll want to disable the “... Joined.” message sent by your client classes because that message will be interpreted as the username.

8. [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.

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.