Louis Keith 11-17-20

Alex Cohen CS 242 Advanced Programming Concepts in Java

Project Part 4 Submission Report

***In your write-up explain why you need two new classes for threading.***

The server and client themselves do not implement runnable and run in the main threads, while the listener classes on each side need their own threads. On the server side, there is a single server main thread running, but there needs to be another thread to listen to each client connected. On the client side, the main thread deals with talking to the user while the client listener is dedicated to communicating with the server.

***In your write-up, explain why there should be a separate class to receive data from the***

***server and print it, and the client only gets data from the user and sends it to the server.***

***Also, why is the class called a ‘listener’?***

The client side listener exists to ‘listen’ to the server and print whatever it receives, that’s its whole job. The main client thread itself is dedicated to actually getting the input from the user and sending it to the server. These are separate classes so that the listener can always be listening for input from other clients while the main client thread is handling its own io. If this wasn’t the case, then the client would only listen for input from the server in specific windows in between prompting the user.

***In your write-up, explain why you need a separate thread for each client, and why you***

***cannot handle all clients in the main server thread. Conceptually, why is the listener***

***class ‘ClientSideServerListener’ different from the class ‘ServerSideClientIO’?***

This is the same deal as on the client side. For each client, there should be an associated listener on the server side that is always available to receive data. If it was done entirely within the main server thread, then it would have to do something like iterate through all of the clients and only listen for a moment before going to the next. This would cause a lot of problems.

***In your write-up, explain why the broadcast() and remove() methods are synchronized.***

***You may find it easier to answer this question after completing all programming.***

Those methods are synchronized because they both access the same data. However, remove() is changing that data. Synchronization eliminates any chance of this causing problems.

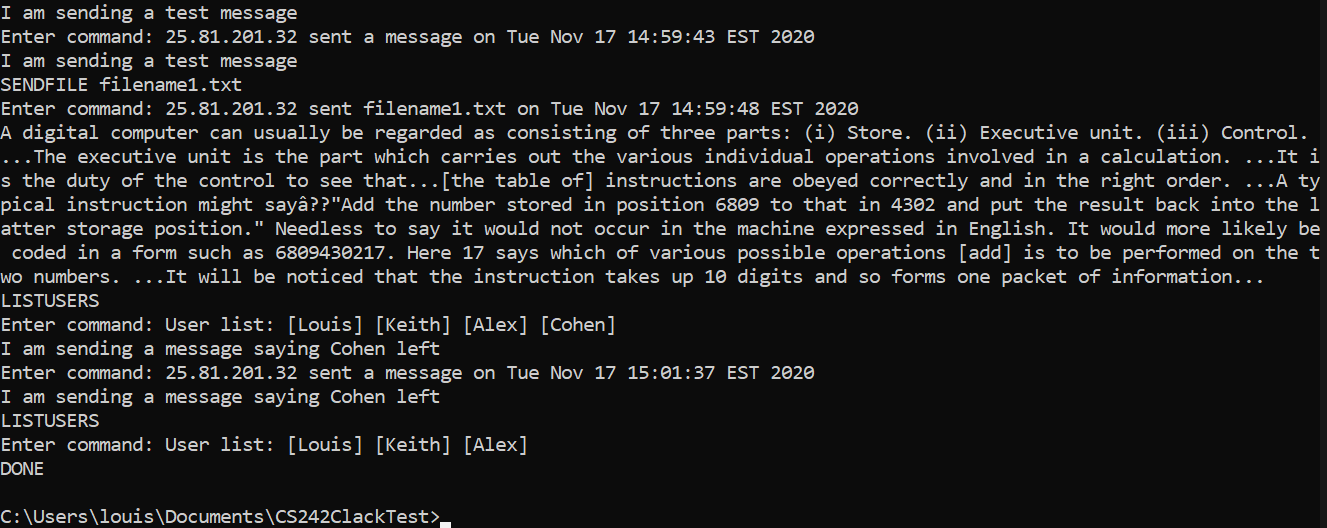
***In your write-up, discuss all new methods and new code in existing***

***methods that you wrote to handle LISTUSERS.***

This was done in the easier way of doing things because we were already so far behind on turning this in. The code was modified so that immediately upon connecting, the client sends a message to the server containing its user name, this message isn’t printed but it allows the server side io listener to locally store the user name of the client it is talking to. There is also an associated getter method for this data so the server itself can access it. In the server, there is a method getUsers() that builds and returns a single string by going through the list of clients and appending each one. The receiveData() method in the server side client io listener was also modified to replace the data it received with a new message containing this user list string, but unencrypted because the server doesn’t store the key. Finally, the printData() method in the client knows that this particular type of message will not be encrypted and makes sure not to attempt to unencrypt it.

***Testing***

Testing went remarkably well. We did not try with three different computers because we didn’t have access to that many, but we did connect as many as four clients at once by using separate terminal windows on the same machines. One of us ran the server and then both of us each connected with two clients as well, so testing should be sufficient. Here is a screenshot of some of our output demonstrating that every command works as it should.

******

There are either two or three problems, depending. The first and most obvious is that the client prompts for the next command before it receives the data from the previous. This doesn’t affect the operation but looks weird. It could probably be fixed with a semaphore type thing, but after some testing this was far from trivial to fix. After speaking to the professor, she said not to worry about it. The second was discovered while testing, it is supposed to display the username and instead chooses to display the hostname of the user. This is a very minor issue and will be fixed for part 5 but we wanted to turn this in as soon as possible. The third issue is completely undetectable while running the program but isn’t following best coding practices. The DONE command could only work properly if we used System.exit(). This is far from ideal, but it works perfectly, and no other solution was forthcoming. Still wanted to mention it.