Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-module-5-project-milestone-1/grade/kh465

Course: IT114-005-F2024
Assigment: [IT114] Module 5 Project Milestone 1
Student: Keven H. (kh465)

Submissions:

Submission Selection

1 Submission [submitted] 10/21/2024 8:22:24 PM

Instructions

COLLAPSE

Overview Video: https://youtu.be/A2yDMS9TS1o

1. Create a new branch called Milestone1
2. At the root of your repository create a folder called Project if one doesn't exist yet
    1. You will be updating this folder with new code as you do milestones
    2. You won't be creating separate folders for milestones; milestones are just branches
3. Copy in the code from Sockets Part 5 into the Project folder (just the files)
    2. https://github.com/MattToegel/IT114/tree/M24-Sockets-Part5
4. Fix the package references at the top of each file (these are the only edits you should do at this point)
5. Git add/commit the baseline and push it to github
6. Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
7. Ensure the sample is working and fill in the below deliverables 1. Note: Don't forget the client commands are /name and /connect
8. Generate the output file once done and add it to your local repository
9. Git add/commit/push all changes
10. Complete the pull request merge from the step in the beginning
11. Locally checkout main
12. git pull origin main

Branch name: Milestone1

Group

100%

Group: Start Up
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Start Up
Task #1: Start Up
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:
Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

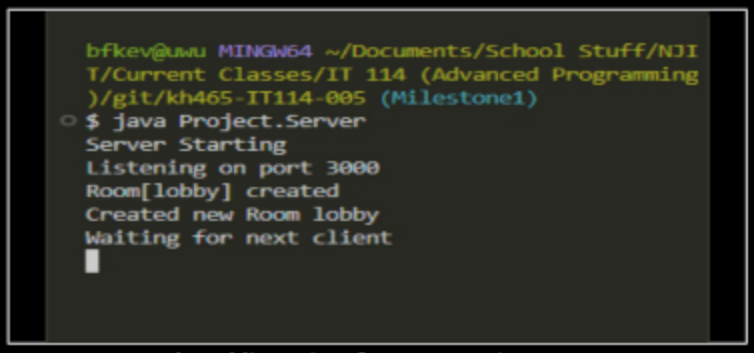
100%

Group: Start Up
Task #1: Start Up
Sub Task #1: Show the Server starting via command line and listening for connections

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server started and listening for connections.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up
Task #1: Start Up
Sub Task #2: Show the Server Code that listens for connections

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Server code that listens for connections, then starts the client.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code related to starting up and waiting for connections

Response:

The method start starts the server, sets the port to 3000 and has a try-with-resources that creates a ServerSocket that waits for a user to connect, then calls the accept method on serverSocket when a client connects. This then creates a ServerThread named sClient which is assigned to the connecting client, and calls its start method.

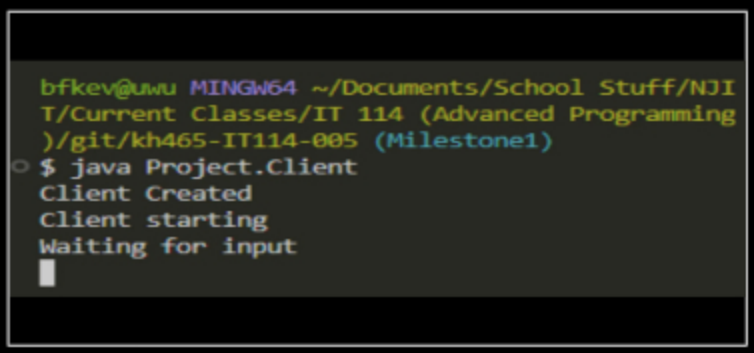
Sub-Task 100%

Group: Start Up
Task #1: Start Up
Sub Task #3: Show the Client starting via command line

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Client started.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Start Up
Task #1: Start Up
Sub Task #4: Show the Client Code that prepares the client and waits for user input

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
42      private Client() {
43          System.out.println(x:"Client Created");
44          myData = new ClientData();
```

Code waiting for user input, either a message or /connect command.

Code that creates client and creates the object ClientData assigned to myData.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow leading up to and including waiting for user input

Response:

Client creates a new instance of the object ClientData and assigns it to myData. Following the code relating to client creation and server connection, the method start() creates a CompletableFuture that runs on a thread, the method listenToInput() awaits the users' input. Scanner si is created and String line takes the input, and while not a client command and not connected. processClientCommand handles the connection, calling isConnection to match the command and ensuring the users' name is not null or 0 characters long.

End of Task 1

Task

100%

Group: Start Up
Task #2: Connecting
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

100%

Group: Start Up
Task #2: Connecting

Sub Task #1: Show 3 Clients connecting to the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Three clients connected to server. Server terminal is denoted by the white star.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Start Up

Task #2: Connecting

Sub Task #2: Show the code related to Clients connecting to the Server (including the two needed commands)

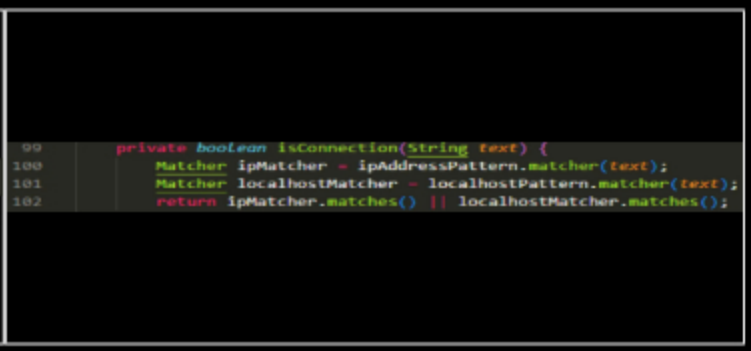
Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
final Pattern ipAddressPattern = Pattern
        .compile(regex:"/connect\\s+(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}:\\d{1,5})");
final Pattern localhostPattern = Pattern.compile(regex:"/connect\\s+(localhost:\\d{1,5})");
```

Code for matching inputted pattern to either an IP address or a localhost address.



```
99      private boolean isConnection(String text) {
100         Matcher ipMatcher = ipAddressPattern.matcher(text);
101         Matcher localhostMatcher = localhostPattern.matcher(text);
102         return ipMatcher.matches() || localhostMatcher.matches();
```

Code using Matcher to handle /connect to either an IP connection or localhost connection.

```
if (isConnection(text)) {
    if (mydata.getClientName() == null || mydata.getClientName().length() == 0) {
        System.out.println(Text.colorize(text:"Name must be set first via /name command", Color.RED));
        return true;
```

Code to process client commands, including connection.

Will fail if clientName is null or 0 chars long.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow

Response:

When the user tries to issue a command to connect to the server, the input is checked against isConnection. Pattern will match that input to either an IP address or a localhost address. From here, if it was either an IP address or localhost address, an if statement checks if the clientName is either null or 0 characters long. If it is, the user must input a /name command, then attempt to /connect again.

End of Task 2

End of Group: Start Up
Task Status: 2/2

Group

100%

Group: Communication
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Communication
Task #1: Communication
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

100%

Group: Communication
Task #1: Communication
Sub Task #1: Show each Client sending and receiving messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Each client sending and receiving data. From left to right: client 1, client 2, client 3, server.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Communication
Task #1: Communication
Sub Task #2: Show the code related to the Client-side of getting a user message and sending it over the socket

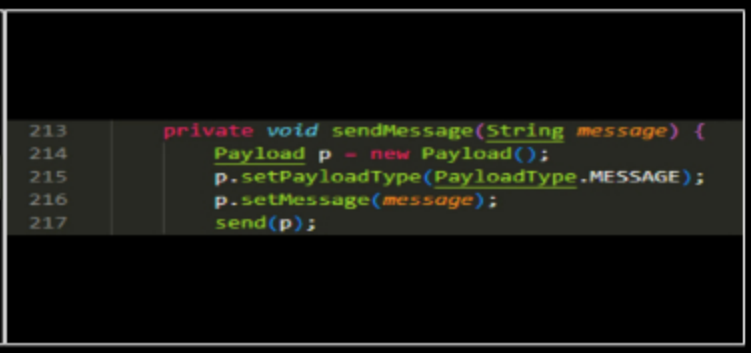
Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
391                 case PayloadType.MESSAGE: // displays a received message
392                     processMessage(payload.getClientId(), payload.getMessage());
393                     break;
```

Receiving user messages from the server.



```
213     private void sendMessage(String message) {
214         Payload p = new Payload();
215         p.setPayloadType(PayloadType.MESSAGE);
216         p.setMessage(message);
217         send(p);
```

Client sending messages to server.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

For a client to send a message, the method listenToInput has a Scanner si that takes user input and stores it in String line that checks if the message the user is attempting to send is a command. If it isn't, then line is passed as an argument to the method sendMessage, which is handled in the processPayload method. The users' clientID is passed along with their message to the payload type MESSAGE.

Sub-Task 100%

Group: Communication
Task #1: Communication
Sub Task #3: Show the code related to the Server-side receiving the message and relaying it to each connected Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
142     public boolean sendMessage(long senderId, String message) {
143         Payload p = new Payload();
144         p.setClientId(senderId);
145         p.setMessage(message);
146         p.setPayloadType(PayloadType.MESSAGE);
147         return send(p);
```

Code receiving the message from the client and broadcasting it to all connected clients.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

A new Payload, p, is created and sets the clientID and message from the arguments passed along from sendMessage, the payloadType is set to MESSAGE, and the method returns send(p).

Sub-Task

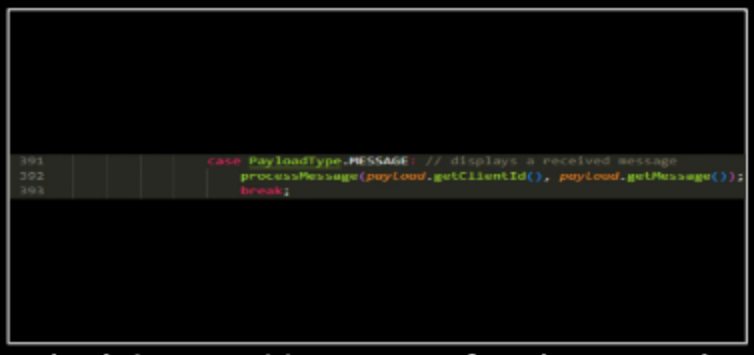
100%

Group: Communication
Task #1: Communication
Sub Task #4: Show the code related to the Client receiving messages from the Server-side and presenting them

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
391                 case PayloadType.MESSAGE: // displays a received message
392                     processMessage(payload.getClientId(), payload.getMessage());
393                     break;
```

Code relating to receiving messages from the server and displaying it to the user.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The method listenToServer() is running while the user is connected to the server. While this method is running, any payloads sent from the server are handled by this method and processed by their relevant method. In the case of receiving messages, the case PayloadType.MESSAGE in processPayload handles the message payload sent by the server, which calls processMessage. processMessage gets the name of the sender, or sets name to "room" if there is no clientID, then formats the message.

End of Task 1

Task

100%

Group: Communication
Task #2: Rooms
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

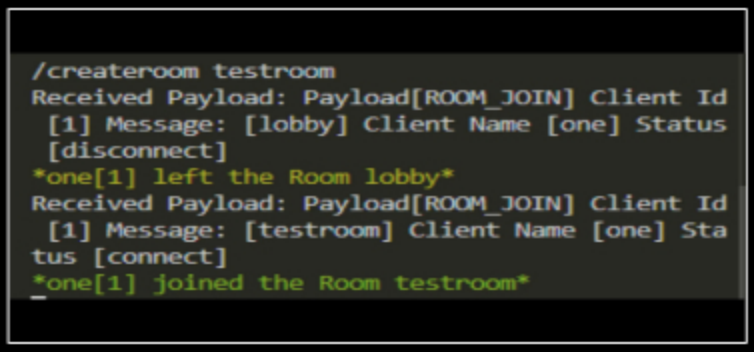
100%

Group: Communication
Task #2: Rooms
Sub Task #1: Show Clients can Create Rooms

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Client one creating a room.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

Group: Communication
Task #2: Rooms

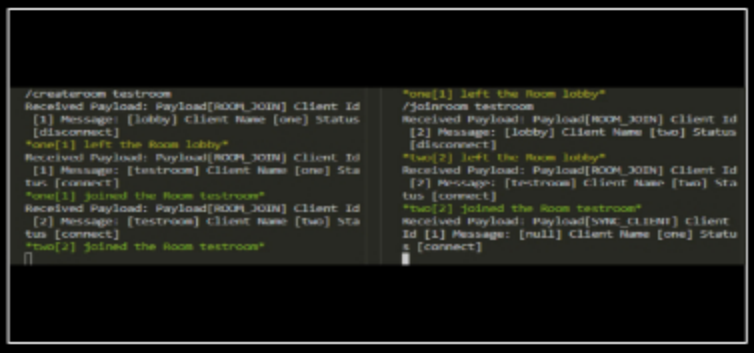
100%

Sub Task #2: Show Clients can Join Rooms (leave/join messages should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Client two joining client one in testroom.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Communication

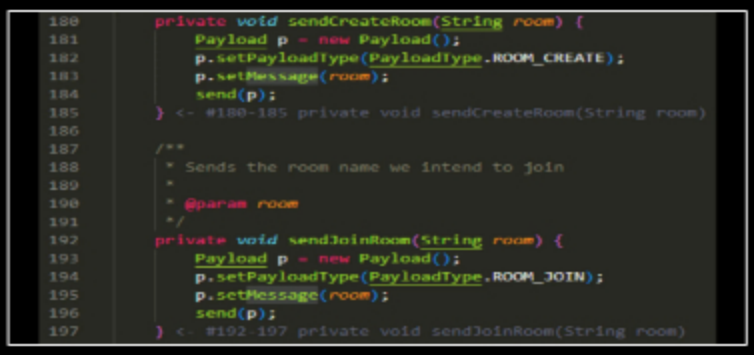
Task #2: Rooms

Sub Task #3: Show the Client code related to the create/join room commands

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
180     private void sendCreateRoom(String room) {
181         Payload p = new Payload();
182         p.setPayloadType(PayloadType.ROOM_CREATE);
183         p.setMessage(room);
184         send(p);
185     } <- #180-185 private void sendCreateRoom(String room)
186
187     /**
188      * Sends the room name we intend to join
189      *
190      * @param room
191      */
192     private void sendJoinRoom(String room) {
193         Payload p = new Payload();
194         p.setPayloadType(PayloadType.ROOM_JOIN);
195         p.setMessage(room);
196         send(p);
197     } <- #192-197 private void sendJoinRoom(String room)
```

Code relating to creating and joining rooms.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

When a user creates a room, a payload of ROOM_CREATE is sent. This checks to see if a room with the same name exists, and if it doesn't then a new room is created with that name, and the user who created the room is placed into it. If another user wants to join a room, joinRoom checks to see if the room exists. If it does, the user is placed into the newly joined room and removed from their prior room.

Sub-Task

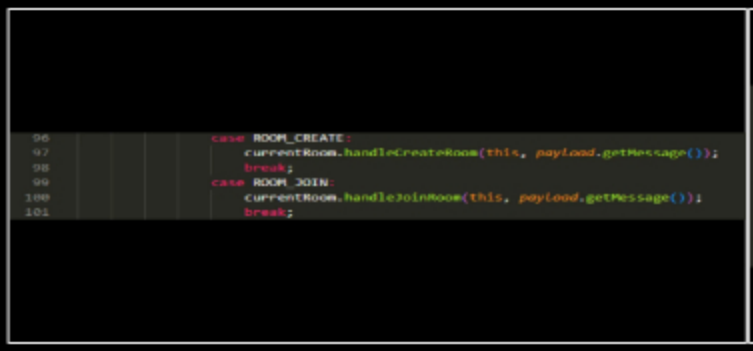
100%

Group: Communication
Task #2: Rooms
Sub Task #4: Show the ServerThread/Room code handling the create/join process

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Code in ServerThread calling handleCreateRoom/handleJoinRoom.

Code in Room telling the Server INSTANCE to create/join rooms.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

When a client issues the payload to create/join a room, this payload is handled by processPayload in ServerThread first, which tells Room to handle the commands. Depending on the command issued, Room issues a command to the Server INSTANCE to create a room with specified name (if it does not already exist), or join the room (if it exists).

Sub-Task

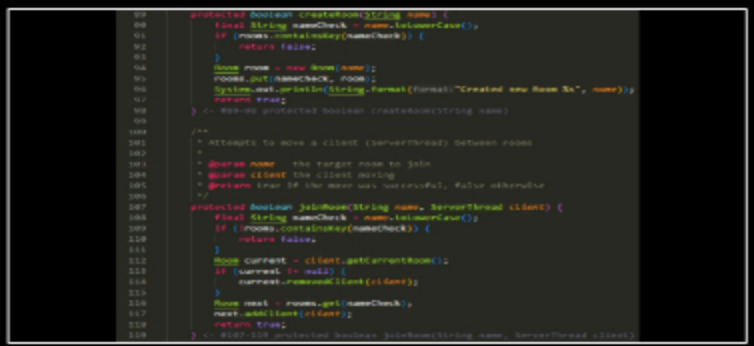
100%

Group: Communication
Task #2: Rooms
Sub Task #5: Show the Server code for handling the create/join process

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server code for handling the create/join commands.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

createRoom passes the argument name and checks if the room already exists. If it does, then it returns false and does not create the room. Otherwise, a new room is created with the name passed along and placed into a ConcurrentHashMap, where it can be joined. joinRoom checks to see if the room exists, and if it does not, returns false and doesn't join the room. Otherwise, it removes the user from their current room, gets the name of the room being joined, and adds the client into the room, and returns true.

Sub-Task

100%

Group: Communication

Task #2: Rooms

Sub Task #6: Show that Client messages are constrained to the Room (clients in different Rooms can't talk to each other)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client one and two talking in testroom, while three does not receive the messages. Rightmost terminal is server.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain why/how it works this way

Response:

Client one and two are in the room testroom, which is a separate room from lobby (which is technically a room, this is the room that all clients join into upon connection). Messages are handled via Server, but processed through Room, which actually displays the messages to users. This is why clients connected to the same room can see each others' messages, but clients in other lobbies cannot.

End of Task 2

End of Group: Communication

Task Status: 2/2

Group

100%

Group: Disconnecting/Termination
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Weight: ~100%
Points: ~3.00

COLLAPSE

Details:
Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

Sub-Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #1: Show Clients gracefully disconnecting (should not crash Server or other Clients)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client three disconnecting from the server, while the server, client one & two remain connected.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

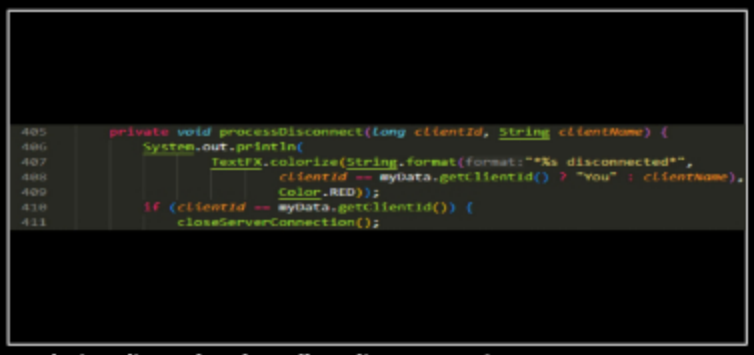
100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #2: Show the code related to Clients disconnecting

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
405     private void processDisconnect(Long clientId, String clientName) {
406         System.out.println(
407                 TextFX.colorize(String.format(format:"%s disconnected*",
408                         clientId == myData.getClientId() ? "You" : clientName),
409                         Color.RED));
410         if (clientId == myData.getClientId()) {
411             closeServerConnection();
```

Code in Client that handles disconnecting.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

When a client sends a disconnect command, the processClientCommand switch case for commands handles the slash command, which calls the sendDisconnect method. In the sendDisconnect method, the users' clientID is checked to ensure that the correct user is the one requesting to be disconnected. If it's the correct user, closeServerConnection() is called to remove the user from the server.

Sub-Task

100%

Group: Disconnecting/Termination

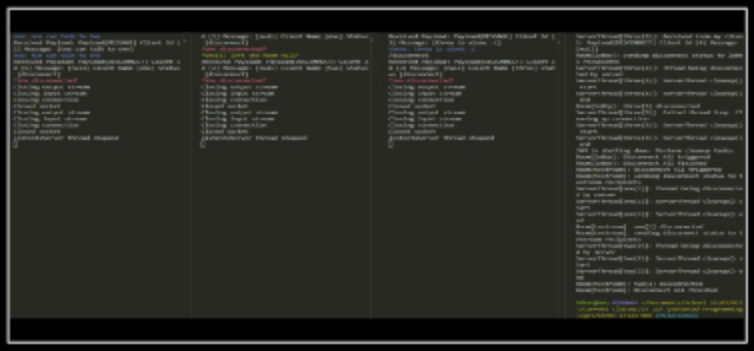
Task #1: Disconnecting

Sub Task #3: Show the Server terminating (Clients should be disconnected but still running)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server gracefully terminating and removing clients from the server. one, two and three are still running, no longer connected.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

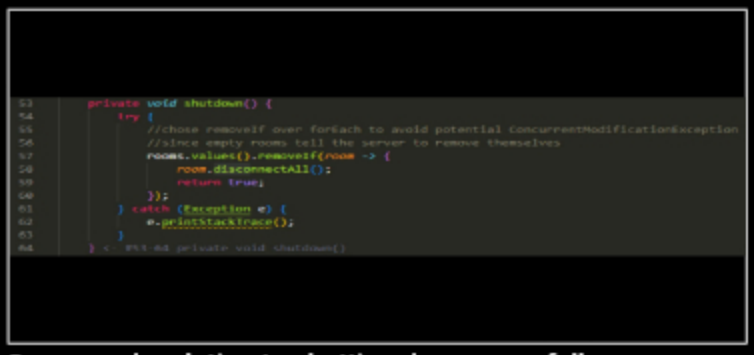
100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #4: Show the Server code related to handling termination

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
53      private void shutdown() {
54          try {
55              //chose removeIf over forEach to avoid potential ConcurrentModificationException
56              //since empty rooms tell the server to remove themselves
57              rooms.values().removeIf(room -> {
58                  room.disconnectAll();
59                  return true;
60              });
61          } catch (Exception e) {
62              e.printStackTrace();
63          }
64      } <  ₹%1-64 private void shutdown()
```

Server code relating to shutting down gracefully.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The shutdown() method in Server tells rooms to disconnect all currently connected users and prints a message to server saying it's performing cleanup tasks.

End of Task 1

End of Group: Disconnecting/Termination
Task Status: 1/1

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Misc
Task #1: Add the pull request link for this branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Task URLs

URL #1
https://github.com/kh465/kh465-IT114-005/pull/12

URL
https://github.com/kh465/kh465-IT114-005/pull/

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Few related sentences about the Project/sockets topics

Task Response Prompt

Response:

No major issues performing this assignment. Further readings and delving into the code will make me more comfortable with the code and making additions to it.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Grab a snippet showing the approximate time involved that clearly shows your repository.

The duration isn't considered for grading, but there should be some time involved.

Task Screenshots

Gallery Style: 2 Columns

4 2 1

WakaTime for 10/21/2024.

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment