Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 1
Student: Colin R. (ctr26)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 7/7/2025 8:42:46 PM
Updated: 7/7/2025 11:32:09 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/grading/ctr26
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/view/ctr26

Instructions

- Overview Link: https://youtu.be/9dZPEwi76ak

1. Refer to Milestone1 of any of these docs:
    2. Rock Paper Scissors
    3. Basic Battleship
    4. Hangman / Word guess
    5. Trivia
    6. Go Fish
    7. Pictionary / Drawing
2. Ensure you read all instructions and objectives before starting.
3. Ensure you've gone through each lesson related to this Milestone
4. Switch to the Milestone1 branch
    1. `git checkout Milestone1` (ensure proper starting branch)
    2. `git pull origin Milestone1` (ensure history is up to date)
5. Copy `Part5` and rename the copy as `Project` (this new folder should be in the root of your repo)
6. Organize the files into their respective packages `Client`, `Common`, `Server`, `Exceptions`
    1. Hint: If it's open, you can refer to the Milestone 2 Prep lesson
7. Fill out the below worksheet
    1. Ensure there's a comment with your UCID, date, and brief summary of the snippet in each screenshot
    2. Since this Milestone was majorly done via lessons, the required comments should be placed in areas of analysis of the requirements in this worksheet. There shouldn't need to be any actual code changes beyond the restructure.
8. Once finished, click "Submit and Export"
9. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`
    3. `git push origin Milestone1`
    4. On Github merge the pull request from `Milestone1` to `main`
10. Upload the same PDF to Canvas
11. Sync Local

1. git checkout main
2. git pull origin main

Section #1: (1 pt.) Feature: Server Can Be Started Via Command Line And Listen To Connections

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of the server started and listening
- Show the relevant snippet of the code that waits for incoming connections

```
Owner@DESKTOP-0F0D1OH MINGW64 /c/git/ctr26-IT114-450 (Milestone1)
$ java Project.Server.Server
Server Starting
Server: Listening on port 3000
Room[lobby]: Created
Server: Created new Room lobby
Server: Waiting for next client
```

Server startup

```
// ctr26 07/07/2025
// This while loop will keep the server listening for clients trying to connect
// Once a connection has been established, the thread will be started
while (isRunning) {
    info(message:"Waiting for next client");
    Socket incomingClient = serverSocket.accept(); // blocking action, waits for a client connection
    info(message:"Client connected");
    // wrap socket in a ServerThread, pass a callback to notify the Server when
    // they're initialized
    ServerThread serverThread = new ServerThread(incomingClient, this::onServerThreadInitialized);
    // start the thread (typically an external entity manages the lifecycle and we
    // don't have the thread start itself)
    serverThread.start();
    // Note: We don't yet add the ServerThread reference to our connectedClients map
} <- #59-70 while (isRunning)
```

Code/Comment

Saved: 7/7/2025 8:50:02 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side waits for and accepts/handles connections

Your Response:

The server runs a while loop in order to remain listening for incoming clients. The server takes the input from the client socket and attempts to create a new server thread with the input, but runs an appropriate catch if the input is invalid.

Saved: 7/7/2025 8:50:02 PM

Section #2: (1 pt.) Feature: Server Should Be Able To Allow More Than One Client To Be Connected At Once

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

Part 1:

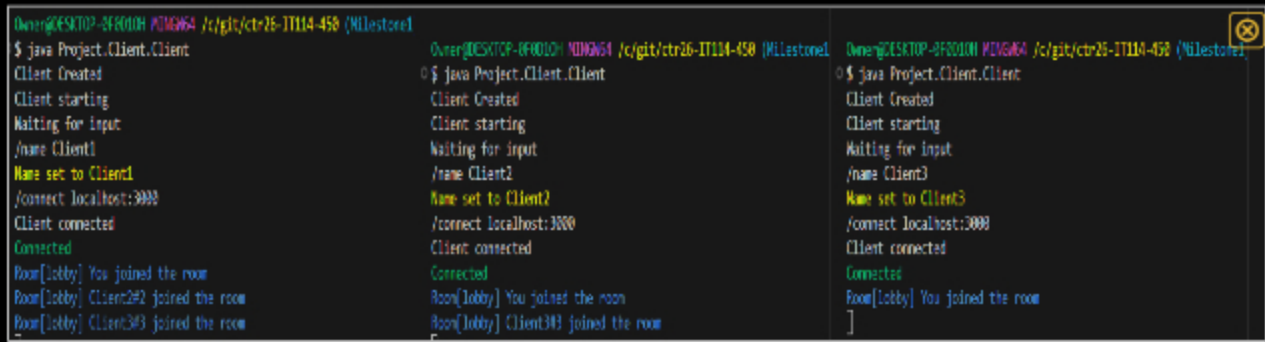
Progress: 100%

Details:

- Show the terminal output of the server receiving multiple connections
- Show at least 3 Clients connected (best to use the split terminal feature)
- Show the relevant snippets of code that handle logic for multiple connections

```
// ctr26 07/07/2025
// Intiallizes each thread with individual client data allowing for multiple connections
ServerThread serverThread = new ServerThread(incomingClient, this::onServerThreadInitialized);
// start the thread (typically an external entity manages the lifecycle and we
// don't have the thread start itself)
serverThread.start();
```

Code/Comment



Client connections

Server output

Saved: 7/7/2025 9:03:52 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles multiple connected clients

Your Response:

By initializing each server thread using the inputted client data, it ensures that each thread has its own respective client. This allows all future thread handling to happen on an individual client basis.

Saved: 7/7/2025 9:03:52 PM

Section #3: (2 pts.) Feature: Server Will Implement The Concept Of Rooms (With The Default Being "l obby")

Progress: 100%

Task #1 (2 pts.) - Evidence

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of rooms being created, joined, and removed (server-side)
- Show the relevant snippets of code that handle room management (create, join, leave, remove) (server-side)

```
// ctr26 07/07/2025
// Upon server startup, the lobby is created as the defaualt room for all clients
createRoom(Room.LOBBY);// create the first room (lobby)
```

Lobby creation

```
// ctr26 07/07/2025
// Checks to see if room already exists before creating new room      You, 48 minutes ago • Unco
protected void createRoom(String name) throws DuplicateRoomException {
    final String nameCheck = name.toLowerCase();
    if (rooms.containsKey(nameCheck)) {
        throw new DuplicateRoomException(String.format(format:"Room %s already exists", name));
    }
    Room room = new Room(name);
    rooms.put(nameCheck, room);
    info(String.format(format:"Created new Room %s", name));
} <- #119-127 protected void createRoom(String name) throws DuplicateRoomEx...
```

Create room

```
// ctr26 07/07/2025
protected void joinRoom(String name, ServerThread client) throws RoomNotFoundException {
    final String nameCheck = name.toLowerCase();
    // Throws exception if the room does not exist
    if (!rooms.containsKey(nameCheck)) {
        throw new RoomNotFoundException(String.format(format:"Room %s wasn't found", name));
    }
    Room currentRoom = client.getCurrentRoom();
    // Removes client from current room in order to add to next room
    if (currentRoom != null) {
        info("Removing client from previous Room " + currentRoom.getName());
        currentRoom.removeClient(client);
    }
    // References the room and runs the command to add the client
    Room next = rooms.get(nameCheck);
    next.addClient(client);
} <- #138-153 protected void joinRoom(String name, ServerThread client) thr...
```

Join room

```
// ctr26 07/07/2025
// Removes the entered room from the hashmap
protected void removeRoom(Room room) {
    rooms.remove(room.getName().toLowerCase());
    info(String.format(format:"Removed room %s", room.getName()));
}
```

Remove room

Server terminal

```
/createroom Room1
Room[lobby] You left the room
Room[Room1] You joined the room
/leaveroom
Room[Room1] You left the room
Room[lobby] You joined the room
/leaveroom
Room[lobby] You left the room
Room[lobby] You joined the room
/joinroom Room1
Room Room1 doesn't exist
/createroom Room1
Room[lobby] You left the room
Room[Room1] You joined the room
```

Client termina;

Saved: 7/7/2025 9:49:31 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the server-side handles room creation, joining/leaving, and removal

Your Response:

Server contains a hashmap with every created room in it. These room can have clients added and removed. Creating a new room simply initializes a room and stores it in the hashmap. removing a room just removes it from the hashmap. The lobby is created when starting the server.

Saved: 7/7/2025 9:49:31 PM

Section #4: (1 pt.) Feature: Client Can Be Started Via The Command Line

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

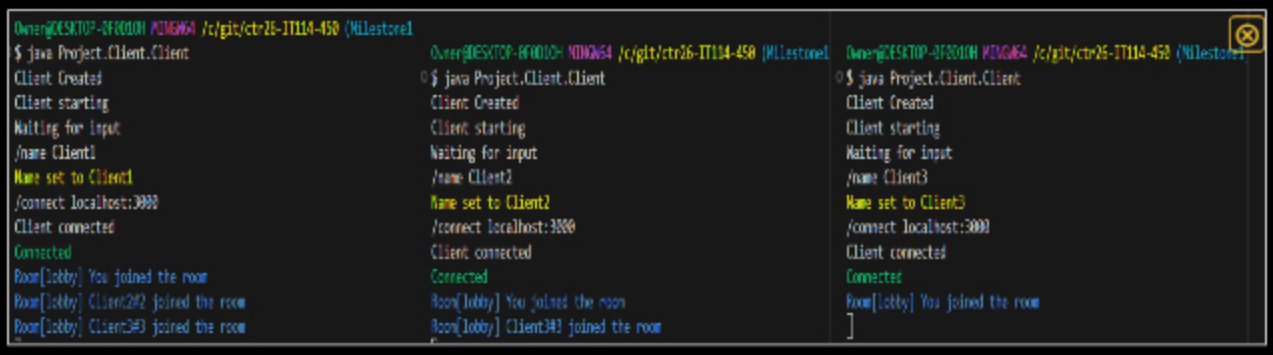
Part 1:

Progress: 100%

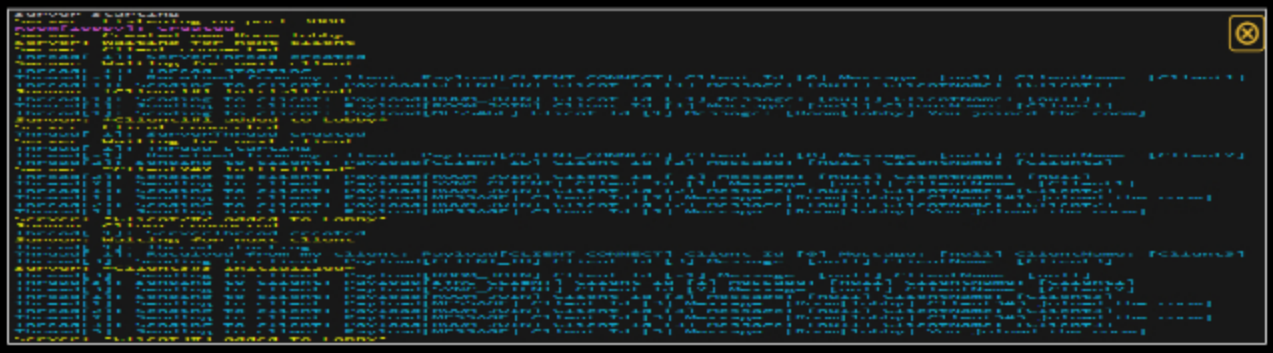
Details:

- Show the terminal output of the /name and /connect commands for each of 3 clients (best to use the split terminal feature)

- Output should show evidence of a successful connection
- Show the relevant snippets of code that handle the processes for /name, /connect, and the confirmation of being fully setup/connected



Client terminal



Server terminal

```java
// ctr26 07/07/2025
// Checks if the client enters the /name command
// Processes the input into the correct format
// Sets the name by using the client's user object
else if (text.startsWith(Command.NAME.command)) {
    text = text.replace(Command.NAME.command, replacement:"").trim();
    if (text == null || text.length() == 0) {
        System.out.println(TextFX.colorize(text:"This command requires a name as an argument", Color.RED));
        return true;
    }
    myUser.setClientName(text);// temporary until we get a response from the server
    System.out.println(TextFX.colorize(String.format(format:"Name set to %s", myUser.getClientName()),
            Color.YELLOW));
    wasCommand = true;
```

/name code/comment

```java
// ctr26 07/07/2025
// Uses isConnection boolean to verify that the input is a valid connection
// Checks if client has name
// Processes input into proper format
// Connects client to server      You, now • Uncommitted changes
if (isConnection("/" + text)) {
    if (myUser.getClientName() == null || myUser.getClientName().isEmpty()) {
        System.out.println(
                TextFX.colorize(text:"Please set your name via /name <name> before connecting", Color.RED));
        return true;
    } <- #129-133 if (myUser.getClientName()     null || myUser.getClientName()....
    // replaces multiple spaces with a single space
    // splits on the space after connect (gives us host and port)
    // splits on : to get host as index 0 and port as index 1
    String[] parts   text.trim().replaceAll(regex:" +", replacement:" ").split(regex:" ")[1].split(regex:":");
    connect(parts[0].trim(), Integer.parseInt(parts[1].trim()));
    sendClientName(myUser.getClientName());// sync follow-up data (handshake)
    wasCommand = true;
} <- #128-141 if (isConnection("/" + text))
```

/connect code/comment

Saved: 7/7/2025 10:01:11 PM

Part 2:

Part 2:

Progress: 100%

Details:

- Briefly explain how the /name and /connect commands work and the code flow that leads to a successful connection for the client

Your Response:

The clients input is processed by determining what command is being used. If the command is /name, the client's User object has its name set to the input. For /connect, the program validates that the input is a connection before actually connecting to the server and creating a serverthread.

Saved: 7/7/2025 10:01:11 PM

Section #5: (2 pts.) Feature: Client Can Create/j oin Rooms

Progress: 100%

Task #1 (2 pts.) - Evidence

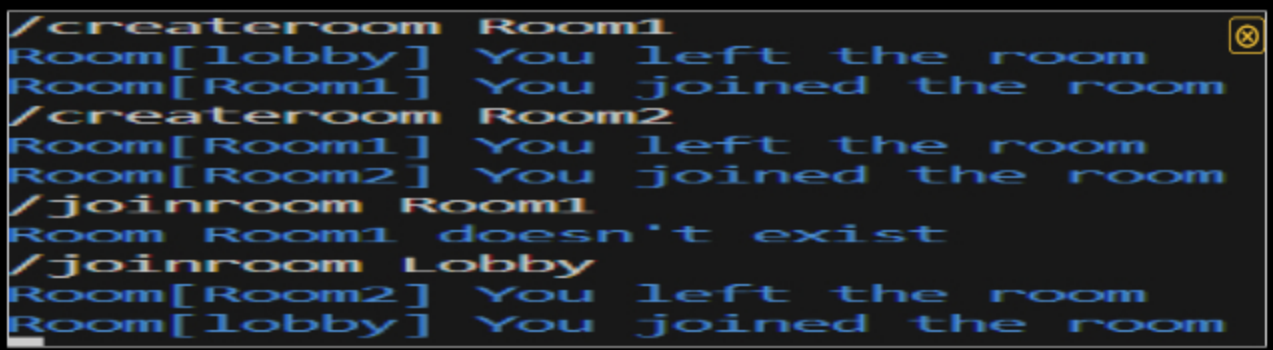
Progress: 100%

Part 1:

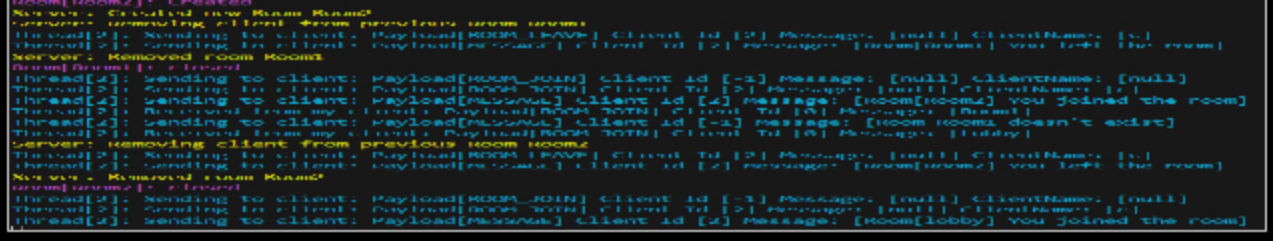
Progress: 100%

Details:

- Show the terminal output of the /createroom and /joinroom
- Output should show evidence of a successful creation/join in both scenarios
- Show the relevant snippets of code that handle the client-side processes for room creation and joining



Client terminal



Server terminal

```
// from 8//8//2025
// Each room command uses the sendRoomAction function to execute command
else if (text.startsWith(Command.CREATE_ROOM.command)) {
    text = text.replace(Command.CREATE_ROOM.command, replacement:"").trim();
    if (text == null || text.length() == 0) {
        System.out.println(TextFX.colorize(text:"This command requires a room name as an argument", Color.RED));
        return true;
    }
    sendRoomAction(text, RoomAction.CREATE);
    wasCommand = true;
} else if (text.startsWith(Command.JOIN_ROOM.command)) {
    text = text.replace(Command.JOIN_ROOM.command, replacement:"").trim();
    if (text == null || text.length() == 0) {
        System.out.println(TextFX.colorize(text:"This command requires a room name as an argument", Color.RED));
        return true;
    }
    sendRoomAction(text, RoomAction.JOIN);
    wasCommand = true;
} else if (text.startsWith(Command.LEAVE_ROOM.command) || text.startsWith(prefix:"leave")) {
    // Note: Accounts for /leave and /leaveroom variants (or anything beginning with
    // /leave)
    sendRoomAction(text, RoomAction.LEAVE);
    wasCommand = true;
} <-- #152-157 else if (text.startsWith(Command.LEAVE_ROOM.command) || text....
```

Command processing

```
// 8//8//2025
// Depending on the room action, a payload is created and sent to the server to execute the acti
private void sendRoomAction(String roomName, RoomAction roomAction) throws IOException {
    Payload payload = new Payload();
    payload.setMessage(roomName);
    switch (roomAction) {
        case RoomAction.CREATE:
            payload.setPayloadType(PayloadType.ROOM_CREATE);
            break;
        case RoomAction.JOIN:
            payload.setPayloadType(PayloadType.ROOM_JOIN);
            break;
        case RoomAction.LEAVE:
            payload.setPayloadType(PayloadType.ROOM_LEAVE);
            break;
        default:
            System.out.println(TextFX.colorize(text:"Invalid room action", Color.RED));
            break;
    } <-- #214 /// switch (roomAction)
    sendToServer(payload);
} <-- #211-229 private void sendRoomAction(String roomName, RoomAction roomA...
```

Payload creation

Saved: 7/7/2025 10:12:27 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the /createroom and /join room commands work and the related code flow for each

Your Response:

The client input is processed into a command and a function is triggered to handle room actions. Inside that function, a payload is created, set to ethe respective room action, and set to the server for processing.

Saved: 7/7/2025 10:12:27 PM

Section #6: (1 pt.) Feature: Client Can Send Messages

Progress: 100%

Task #1 (1 pt.) - Evidence

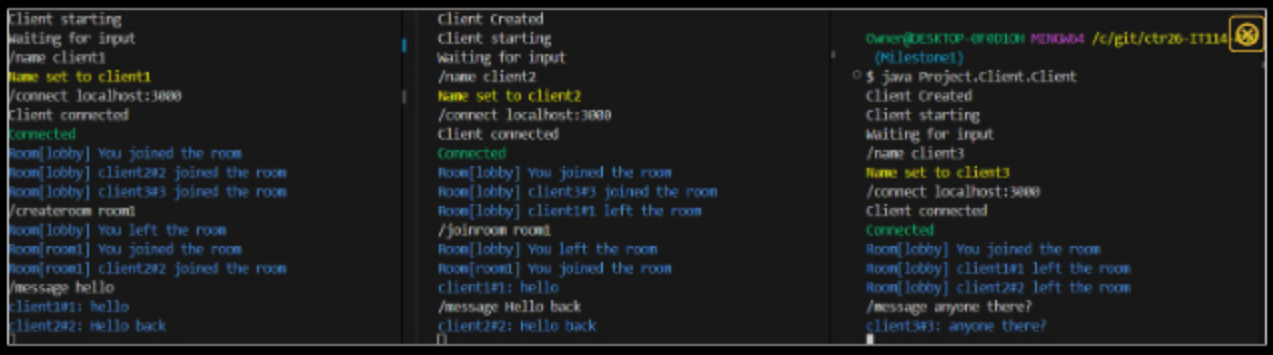
Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the terminal output of a few messages from each of 3 clients
- Include examples of clients grouped into other rooms
- Show the relevant snippets of code that handle the message process from client to server-side and back



Client messages

```
Thread[1]: Received from my client: Payload[MESSAGE] Client Id [0] Message: [hello]
Room[room1]: sending message to 2 recipients: client1#1: hello
Thread[1]: Sending to client: Payload[MESSAGE] Client Id [1] Message: [client1#1: hello]
Thread[2]: Sending to client: Payload[MESSAGE] Client Id [1] Message: [client1#1: hello]
Thread[2]: Received from my client: Payload[MESSAGE] Client Id [0] Message: [Hello back]
Room[room1]: sending message to 2 recipients: client2#2: Hello back
Thread[1]: Sending to client: Payload[MESSAGE] Client Id [2] Message: [client2#2: Hello back]
Thread[2]: Sending to client: Payload[MESSAGE] Client Id [2] Message: [client2#2: Hello back]
Thread[3]: Received from my client: Payload[MESSAGE] Client Id [0] Message: [anyone there?]
Room[lobby]: sending message to 1 recipients: client3#3: anyone there?
Thread[3]: Sending to client: Payload[MESSAGE] Client Id [3] Message: [client3#3: anyone there?]
```

Server terminal

Saved: 7/7/2025 10:37:43 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

The client input is processed into a command with the desired text. This is then used to create a payload that is sent to the server.

Saved: 7/7/2025 10:37:43 PM

Section #7: (1 pt.) Feature: Disconnection

Progress: 100%

Task #1 (1 pt.) - Evidence

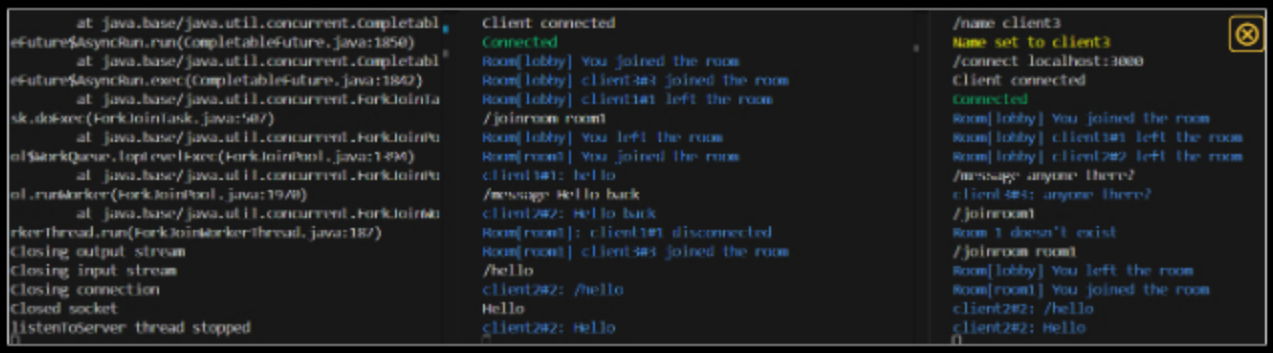
Progress: 100%

Part 1:

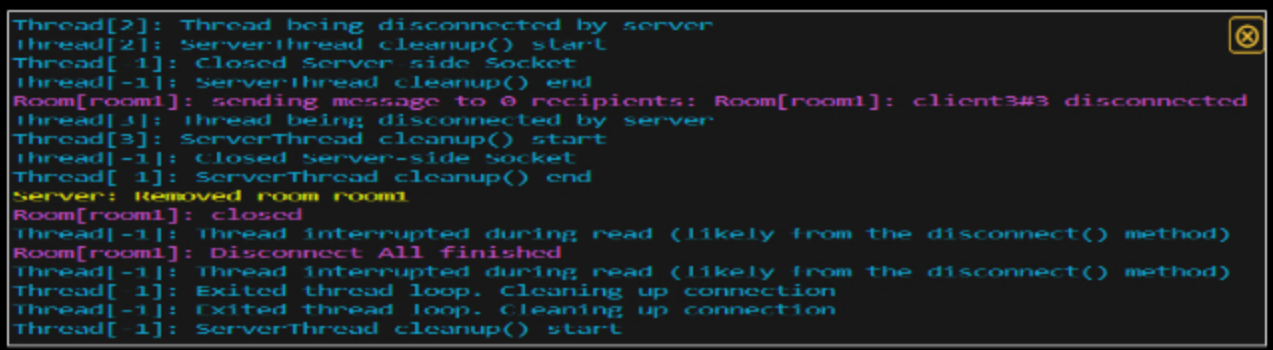
Progress: 100%

Details:

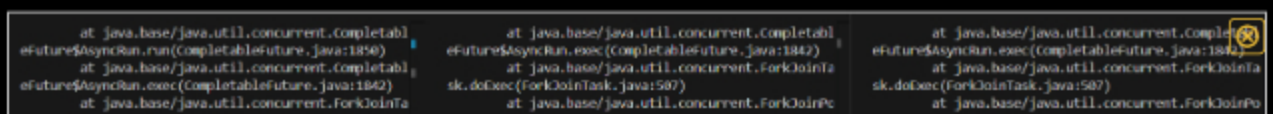
- Show examples of clients disconnecting (server should still be active)
- Show examples of server disconnecting (clients should be active but disconnected)
- Show examples of clients reconnecting when a server is brought back online
- Examples should include relevant messages of the actions occuring
- Show the relevant snippets of code that handle the client-side disconnection process
- Show the relevant snippets of code that handle the server-side termination process

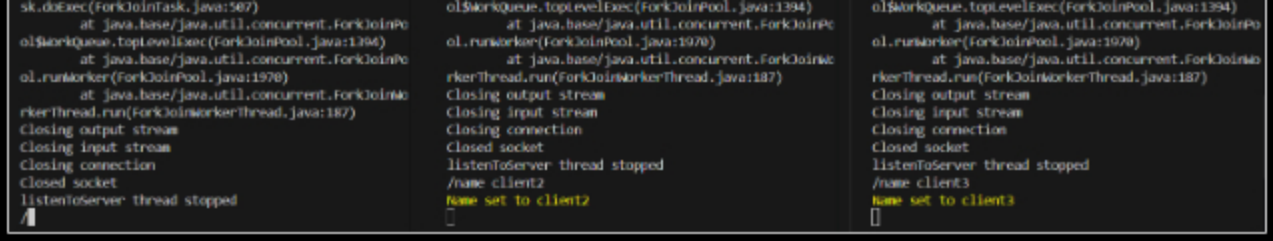


Client disconnect

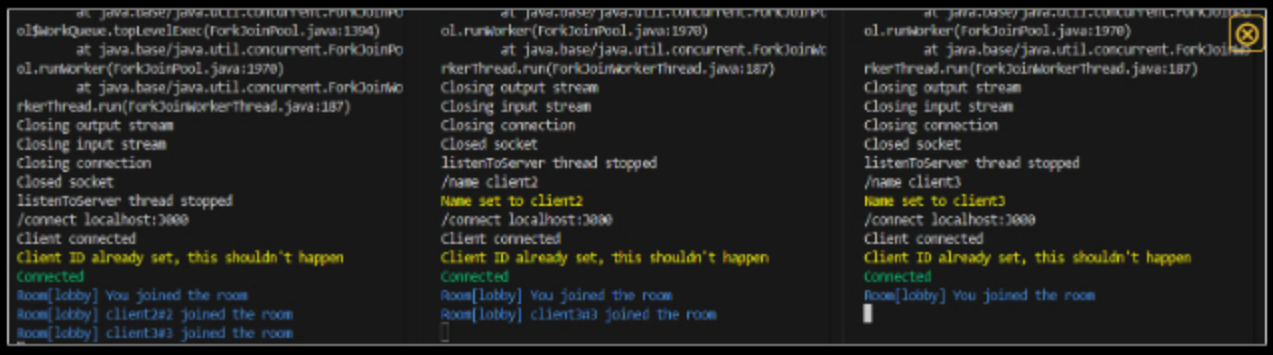


Server disconnect





Server disconnect on clients



Clients reconnecting

Client side disconnect code/comment

```
// ctr26 07/07/2025
// Disconnects every client from every room without killing client program
private void shutdown() {
    try {
        // chose removeIf over forEach to avoid potential
        // ConcurrentModificationException
        // since empty rooms tell the server to remove themselves
        rooms.values().removeIf(room -> {
            room.disconnectAll();
            return true;
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
} <- #40-52 private void shutdown()
```

Server shutdown code/comment

Saved: 7/7/2025 11:21:57 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how both client and server gracefully handle their disconnect/termination logic

Your Response:

Client handles disconnection by just closing the server thread between the client and server. The server handles disconnection by removing all clients from the server's rooms before shutting down.

Saved: 7/7/2025 11:21:57 PM

Section #8: (1 pt.) Misc

Progress: 100%

Task #1 (0.25 pts.) - Show the proper workspace structure with the new Client, Common, Server, and Exceptions packages

Progress: 100%

Workspace 1/2

Workspace 2/2

Saved: 7/7/2025 11:23:32 PM

Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:

From the Commits tab of the Pull Request screenshot the commit history

Commit history

Saved: 7/7/2025 11:26:00 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request (should end in `/pull/#`)

URL #1
https://github.com/ColinRafferty7/ctr26-IT114-450

URL
https://github.com/ColinRafferty7

Saved: 7/7/2025 11:26:00 PM

Task #3 (0.25 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary

Wakatime top

Wakatime bottom

Saved: 7/7/2025 11:27:33 PM

Task #4 (0.25 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

For this assignment, I learned how these connections are handled a lot more in depth. Having to go through and explain each part of the connection allowed me to dissect the code and understand the ins and outs of it. There are still parts that I will have to go back through in the future to be able to get a full grasp of.

Saved: 7/7/2025 11:28:58 PM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The easiest part of this assignment was getting everything set up in order to start fillin out the worksheet. Everything was already done, so it was only a matter of reorganizing everything and setting up the branch and pull request.

Saved: 7/7/2025 11:29:56 PM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

The hardest part of this assignment was getting all of the output screenshots from every test. It was really tedious and the server program produces a lot of outputs, which makes it hard to capture specific lines.

Saved: 7/7/2025 11:32:09 PM