# Submission Worksheet

## Submission Data

**Course:** IT114-450-M2025
**Assignment:** IT114 Milestone 1
**Student:** Aditya D. (ad273)
**Status:** Submitted | **Worksheet Progress:** 96%
**Potential Grade:** 9.50/10.00 (95.00%)
**Received Grade:** 0.00/10.00 (0.00%)
**Started:** 7/7/2025 6:07:02 PM
**Updated:** 7/7/2025 7:22:25 PM
**Grading Link:** https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/grading/ad273
**View Link:** https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/view/ad273

## Instructions

- Overview Link: https://youtu.be/9dZPFwi76ak

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

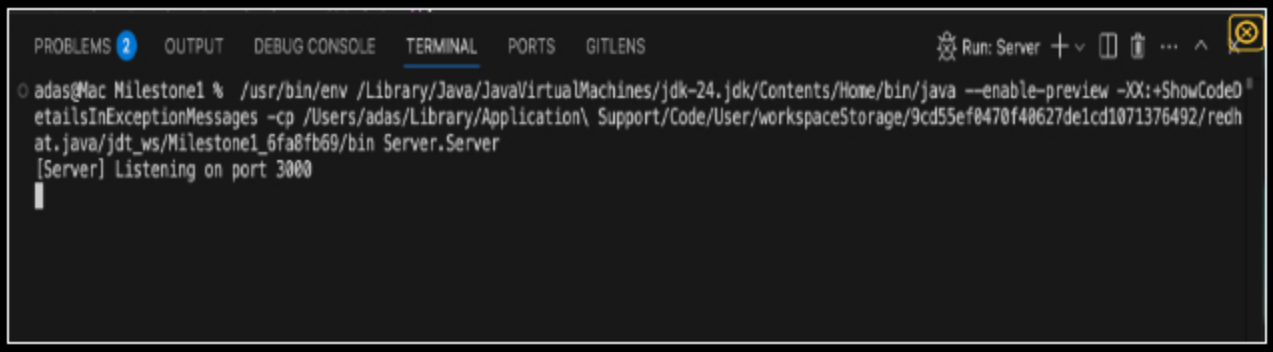
Progress: 100%

### 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




Server Listening to port 3000

```
12
13      //ucid = ad273 7/7/25 - "This loop waits for clients to connect via serverSocket.accept(),
14      //creates a thread for each, and adds them to the lobby."
        Run | Debug
15      public static void main(String[] args) {
16          try (ServerSocket serverSocket = new ServerSocket(PORT)) {
17              System.out.println("[Server] Listening on port " + PORT);
18              while (true) {
19                  Socket clientSocket = serverSocket.accept();
20                  ServerThread clientThread = new ServerThread(clientSocket, lobby);
21                  clients.add(clientThread);
22                  lobby.addClient(clientThread);
23                  clientThread.start();
24              }
25          } catch (IOException e) {
26              e.printStackTrace();
```

Server Code and Comment with UCID

Saved: 7/7/2025 6:10:55 PM

### Part 2:

Progress: 100%

**Details:**

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

Your Response:

The server uses a ServerSocket to listen for incoming client connections on a specific port. It waits for a connection using serverSocket.accept(), which blocks the thread until a client connects. Once a client connects, it returns a Socket object used to communicate with that client. The server then creates a new ServerThread for the client, adds it to the list of active clients, and starts the thread so the client can be handled concurrently.

Saved: 7/7/2025 6:10:55 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

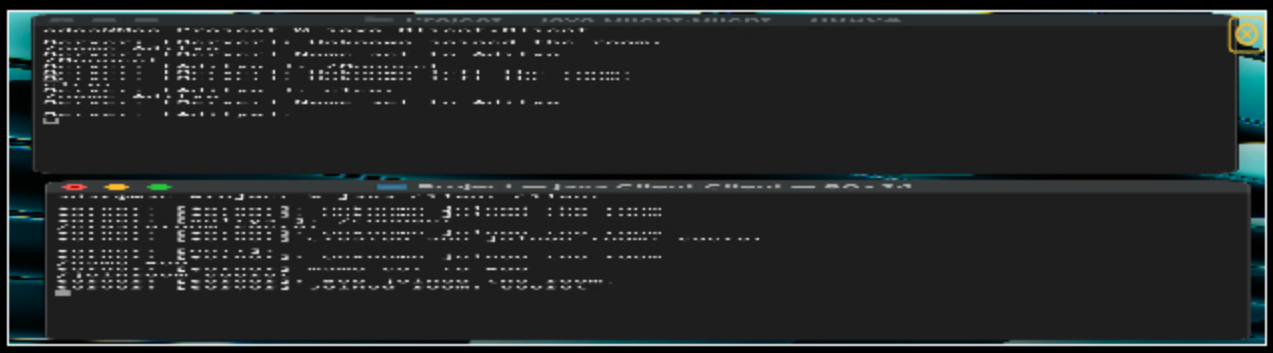
Progress: 100%

### Part 1:

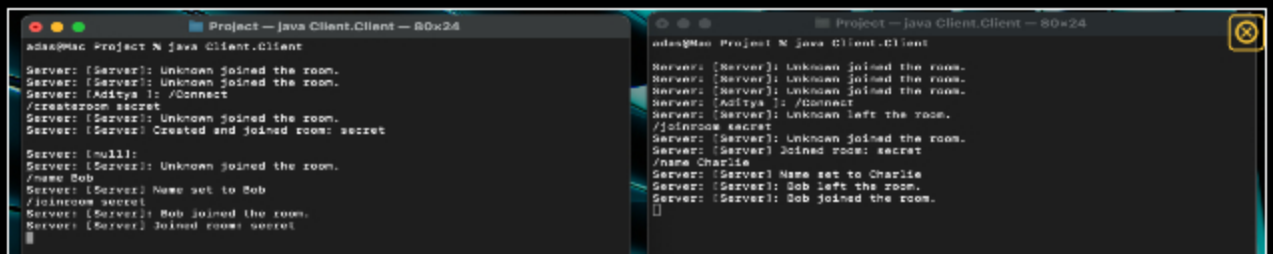
Progress: 100%

**Details:**

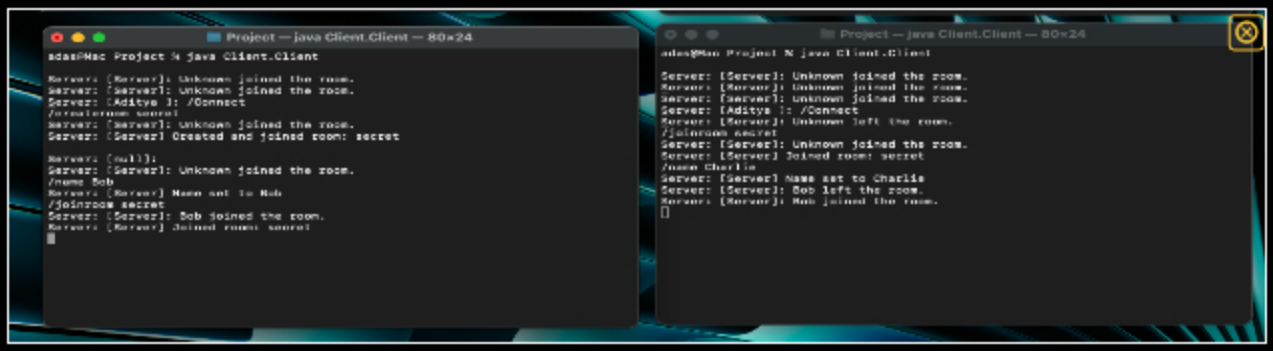
- Show the terminal output of the server receiving multiple connections
- Show at least 3 Clients connected (best to use the split terminal feature)
- Show the relevant snippets of code that handle logic for multiple connections



Terminal Outputs



Terminal outputs



Terminal Outputs

```java
//UCID: ad273 Date:7/7/25
public class ServerThread extends Thread {
    private Socket socket;
    private ObjectInputStream input;
    private ObjectOutputStream output;
    private String clientName;
    private Room currentRoom;

    public ServerThread(Socket socket, Room room) {
        this.socket = socket;
        this.currentRoom = room;
    }

```

This is what lets each client interact independently. Every client has its own ServerThread.

Saved: 7/7/2025 6:35:29 PM

## Part 2:

Progress: 100%

**Details:**

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

Your Response:

The server uses a thread-per-client model where each client is handled by its own ServerThread, allowing multiple clients to be served at the same time independently.

Saved: 7/7/2025 6:35:29 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%

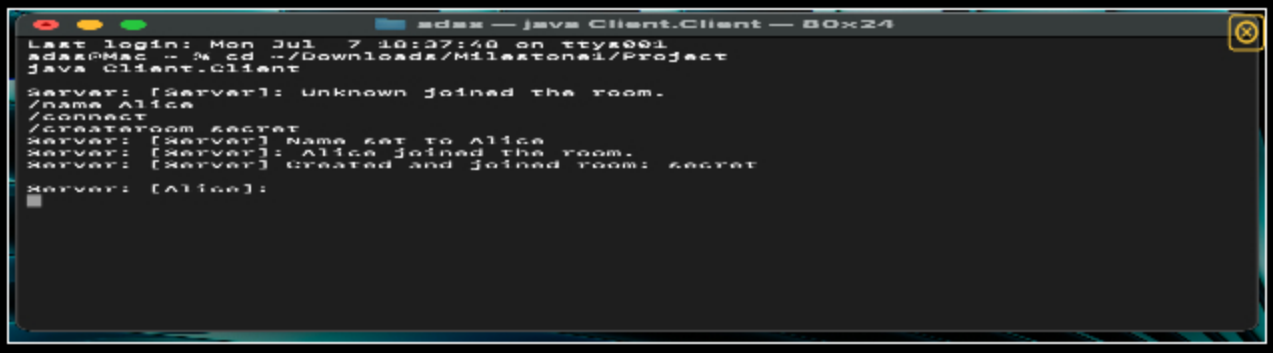
## Part 1:

Progress: 100%

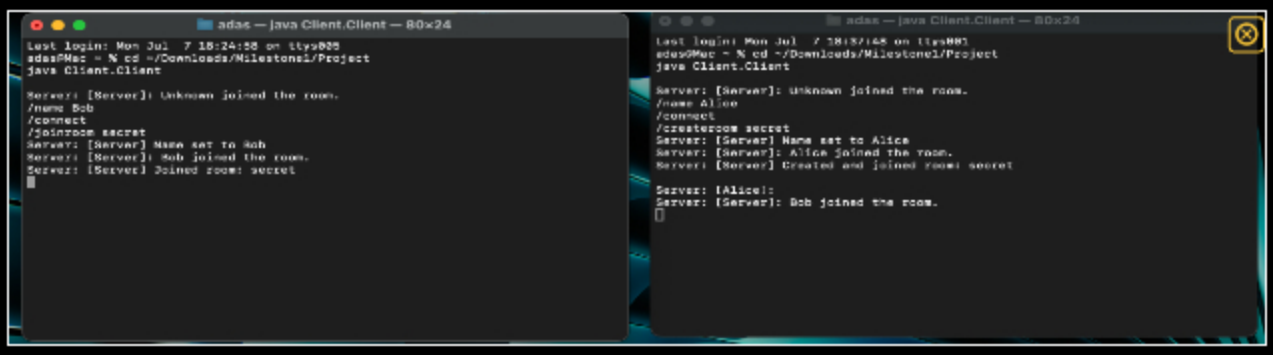
**Details:**

- 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)

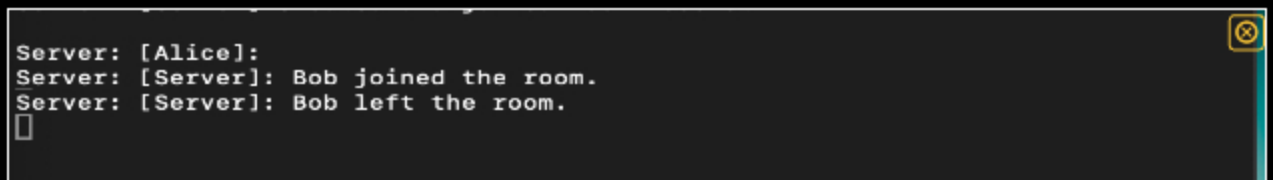

Disconnected from server

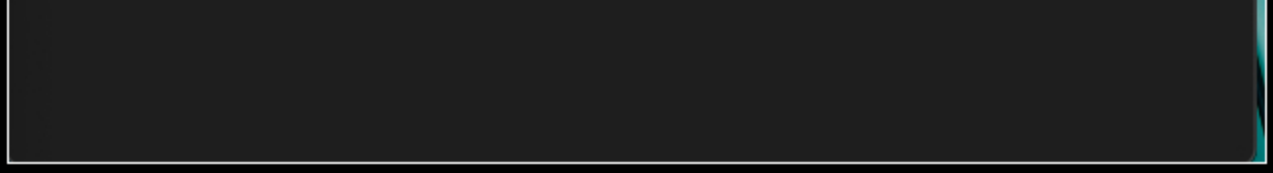



Create




Join





Leave

Saved: 7/7/2025 6:42:25 PM

### Part 2:

Progress: 100%

**Details:**

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

Your Response:

The server manages rooms using a Room class, which keeps track of connected clients. By default, all clients start in a shared "lobby" room.

When a client uses /createroom, the server creates a new Room instance and moves the client into it. When a client uses /joinroom, the server checks if the room exists and transfers the client from their current room into the new one.

During either action, the server calls removeClient(this) on the old room and addClient(this) on the new room to update the membership.

If a client disconnects or leaves, they are also removed from their current room using the same removeClient() method. All these room changes are broadcast to users in the room for awareness.

Saved: 7/7/2025 6:42:25 PM

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

Progress: 100%

### Task #1 ( 1 pt.) - Evidence

Progress: 100%

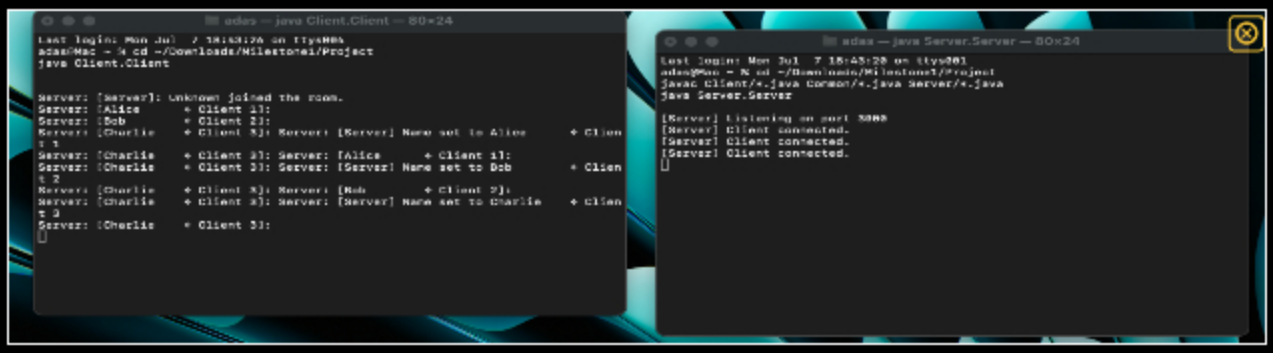
### Part 1:

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




Multiple clients


Handles the client and how it connects

Saved: 7/7/2025 6:49:43 PM

## 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:

When the client types /name Alice, it sends a message to the server with the name "Alice". The server stores that name for the client.

Then when the client types /connect, it tells the server "I'm ready", and the server responds with a confirmation message.

Each client runs in its own terminal and connects separately, but they all join the same server and are able to talk in the same room (lobby by default).

Saved: 7/7/2025 6:49:43 PM

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

Progress: 100%

## Task #1 ( 2 pts.) - Evidence

Progress: 100%

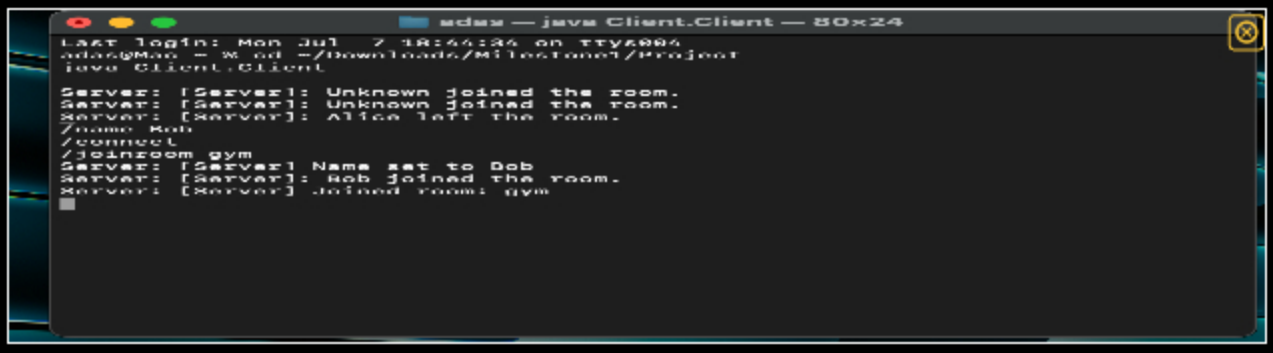
### Part 1:

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 1




Client 2


room updates

```
        //ucid: ad273  Date: 7/7/25
        while (true) {
            String line = scanner.nextLine();
            if (line.startsWith(prefix:"/name ")) {
                Payload p = new Payload(PayloadType.SET_NAME);
                p.setMessage(line.substring(beginIndex:6));
                output.writeObject(p);
            } else if (line.startsWith(prefix:"/connect")) {
                Payload p = new Payload(PayloadType.CONNECT);
                output.writeObject(p);
            } else if (line.startsWith(prefix:"/createroom ")) {
                Payload p = new Payload(PayloadType.CREATE_ROOM);
                p.setMessage(line.substring(beginIndex:12));
                output.writeObject(p);
            } else if (line.startsWith(prefix:"/joinroom ")) {
                Payload p = new Payload(PayloadType.JOIN_ROOM);
                p.setMessage(line.substring(beginIndex:10));
                output.writeObject(p);
```

Create Room and Join Room

Saved: 7/7/2025 6:53:44 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:

When a client types /createroom gym, the client sends a message to the server asking to create a new room called "gym". The server creates the room and moves the client into it.

When another client types /joinroom gym, it tells the server to switch them from their current room to "gym". The server removes them from their old room (like the lobby) and adds them to the new one.

This room switching allows clients to have private or group conversations in separated spaces.

Saved: 7/7/2025 6:53:44 PM

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

Progress: 50%

## Task #1 ( 1 pt.) - Evidence

Progress: 50%

### Part 1:

Progress: 0%

**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


Missing Caption

Saved: 7/7/2025 7:22:25 PM

### Part 2:

Progress: 100%

**Details:**

- Briefly explain how the message code flow works

Your Response:

Client Sends Message When a client types and sends a message, the client-side creates a Payload object (with type MESSAGE) and sends it to the server via the output stream.

Server Receives Message On the server side, the ServerThread listening to that client receives the message payload. It checks the type (PayloadType.MESSAGE).

Server Broadcasts Message to Room The ServerThread then passes the message to its current room (Room object). The Room class has a method that loops through all users in that room and sends the message to each of them.

Other Clients Receive It Each client that's part of that room receives the message from the server input stream and displays it in their terminal.

Saved: 7/7/2025 7:22:25 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

```
/name Bob
/connect
/joinroom gym
Server: [Server] Name set to Bob
Server: [Server]: Bob joined the room.
Server: [Server] Joined room: gym
/exit
Disconnected from server.
adas@Mac Project %
```

Client Exit

```
Server: [Server]: Unknown joined the room.
Server: [Server]: Alice left the room.
Server: [Server]: Bob left the room.
```

Leaving the Room

```
Ctrl + C

java Server.Server

/name Bob
/connect

[Server] Client disconnected or error occurred.
[Server] Client disconnected or error occurred.
[Server] Client connected.
[Server] Client connected.
[Server] Client connected.
```

Reconnection

```
        } else if (line.startsWith(prefix:"/exit")) {
            Payload p = new Payload(PayloadType.DISCONNECT);
            output.writeObject(p);
            break;
        } else {
            Payload p = new Payload(PayloadType.MESSAGE);
            p.setMessage(line);
            output.writeObject(p);
```

```
        }
    }
```

Disconnect Code

```
                    break;
                case DISCONNECT:
                    close();
                    return;
            }
        }
    } catch (Exception e) {
        System.out.println(x:"[Server] Client disconnected or error occurred.");
    } finally {
        close();
    }
}
```

DC from server thread

Saved: 7/7/2025 6:59:37 PM

## Part 2:

Progress: 100%

**Details:**

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

Your Response:

When a client types /exit, it sends a DISCONNECT payload to the server. The server removes that client from their current room and closes the connection cleanly.

If the server is shut down, the clients catch the lost connection and display "Disconnected from server."

Once the server is restarted, clients can reconnect by rerunning the connection commands (/name and /connect), and the server will treat them as new sessions.

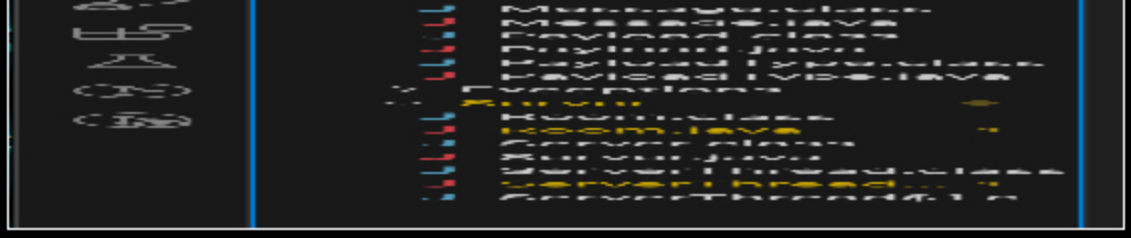
Saved: 7/7/2025 6:59:37 PM

# 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%



Folder Tree

Saved: 7/7/2025 7:00:36 PM

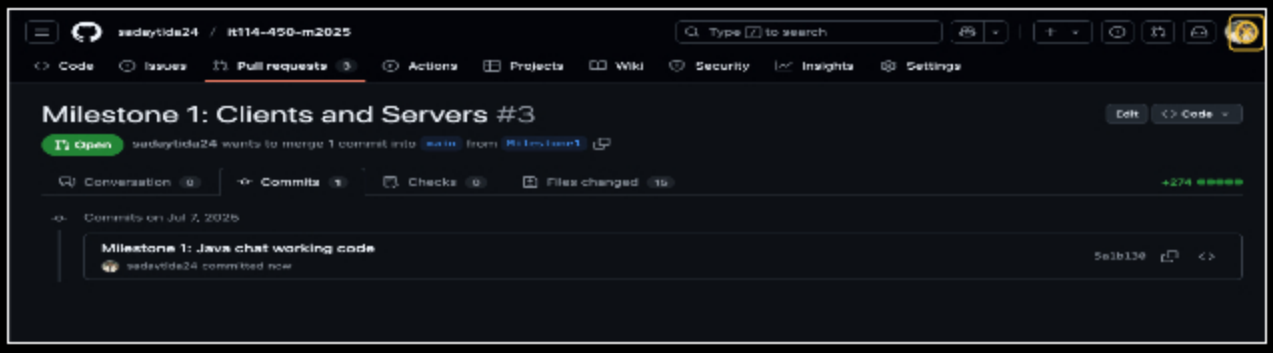
## Task #2 ( 0.25 pts.) - Github Details

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 7:13:16 PM

### Part 2:

Progress: 100%

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

URL #1
https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/Milestone%201:%20Clients%20and%20Servers%20#3

URL
https://learn.ethereallab.app/assi

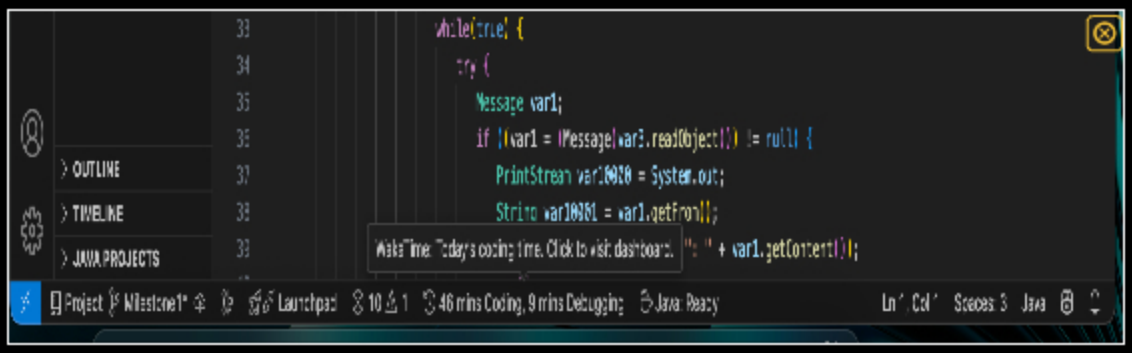
Saved: 7/7/2025 7:13:16 PM

# 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

Saved: 7/7/2025 7:16:14 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:

I learned how to build a multi-threaded Java client-server application that allows real-time communication between multiple clients. I gained hands-on experience with sockets, threading, payload structures, and how to organize a project using packages like Client, Server, Common, and Exceptions. I also learned how to implement chat room logic, properly manage server-client connections, and handle user commands from the terminal.

Saved: 7/7/2025 7:16:52 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 was understanding the basic client-server communication flow using Java sockets. Once the initial connection setup was working, sending and receiving messages between clients and the server became more straightforward. Structuring the project into packages was also relatively simple and helped keep everything organized.

Saved: 7/7/2025 7:17:05 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 was debugging errors related to threading and connection handling, especially when trying to support multiple clients at the same time. I also ran into issues with pushing the working project to GitHub and creating a clean pull request, which took some troubleshooting in the terminal. Making sure the reconnection and room logic worked correctly took extra effort as well.

Saved: 7/7/2025 7:17:15 PM