Submission Worksheet

Submission Data

Course: IT114-005-F2025
Assignment: IT114 Milestone 1
Student: Nilkanth D. (nhd5)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 11/3/2025 11:05:03 PM
Updated: 11/4/2025 12:57:43 AM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-1/grading/nhd5
View Link: https://learn.ethereallab.app/assignment/v3/IT114-005-F2025/it114-milestone-1/view/nhd5

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

```java
public enum Server {
    INSTANCE; // Singleton instance

    private int port = 3000;
    // connected clients
    // Use ConcurrentHashMap for thread-safe client management
    // The key is the unique Room name and the Room is the instance
    private final ConcurrentHashMap<String, Room> rooms = new ConcurrentHashMap<>();
    private boolean isRunning = true;
    private long nextClientId = 0;

    private void info(String message) {
        System.out.println(TextFX.colorize(String.format("Server: %s", message), Color.YELLOW));
    }

    private Server() {
```

server connection

Saved: 11/3/2025 11:06:21 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 object that continuously listens for incoming client connections on port 3000. When a client connects, the accept() method creates a new socket for that client, allowing communication to begin. Each client is then handled on a separate thread so the server can manage multiple connections at the same time without stopping.

Saved: 11/2/2025 11:06:21 PM

Saved: 11/3/2025 11:06:21 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

more than one client

Saved: 11/4/2025 12:26:22 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server uses a multithreading approach to handle multiple clients at once. When a new client connects, the main Server creates a new ServerThread (extending BaseServerThread) to manage that specific client's communication. Each thread runs independently, allowing several clients to connect, send, and receive data simultaneously without interrupting one another.

Saved: 11/4/2025 12:26:22 AM

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)

```
56    public synchronized void createRoom(String name) throws DuplicateRoomException {
57        String lower = name.toLowerCase();
58        if (rooms.containsKey(lower)) {
59            throw new DuplicateRoomException("Room " + name + " already exists");
60        }
61        Room newRoom = new Room(name);
62        rooms.put(lower, newRoom);
63        System.out.println("Created Room: " + name);
64    }
```

server.java

room.java

Saved: 11/4/2025 12:29:46 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server uses a ConcurrentHashMap to keep track of all active rooms, with "lobby" created by default when the server starts. When a client uses /createroom, the server adds a new Room object to this map. The /joinroom command moves the client's thread from its current room to the selected one, while /leaveroom returns the client to the lobby. If a room becomes empty, the server automatically closes and removes it from memory to keep things organized and efficient.

Saved: 11/4/2025 12:29:46 AM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

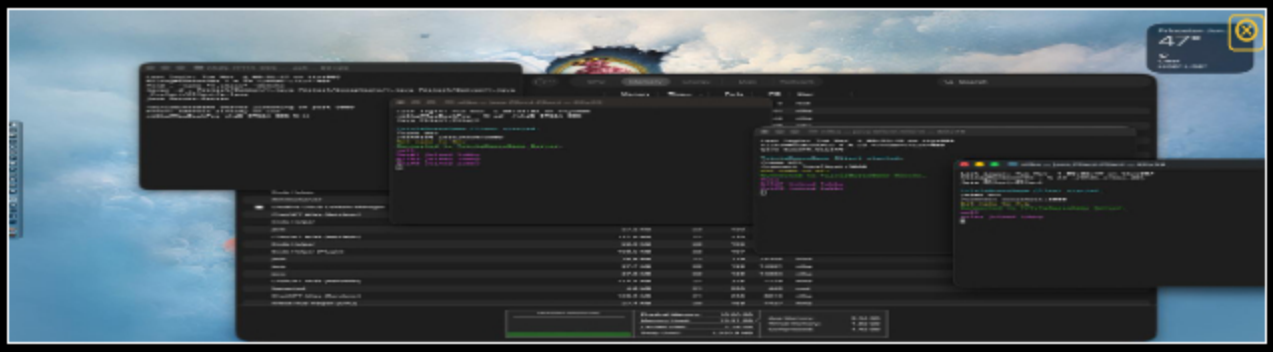
Progress: 100%

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



All three clients connected successfully to the server using /name and /connect commands via command line.

Saved: 11/4/2025 12:36:09 AM

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 enters the /name command, the program updates the user's display name locally by storing it in a User object. The /connect command then creates a socket connection to the server using the provided address and port, sets up input/output streams, and starts a background thread to listen for messages. Once connected, the client automatically sends its name to the server, confirming a successful connection and registration.

Saved: 11/4/2025 12:36:09 AM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

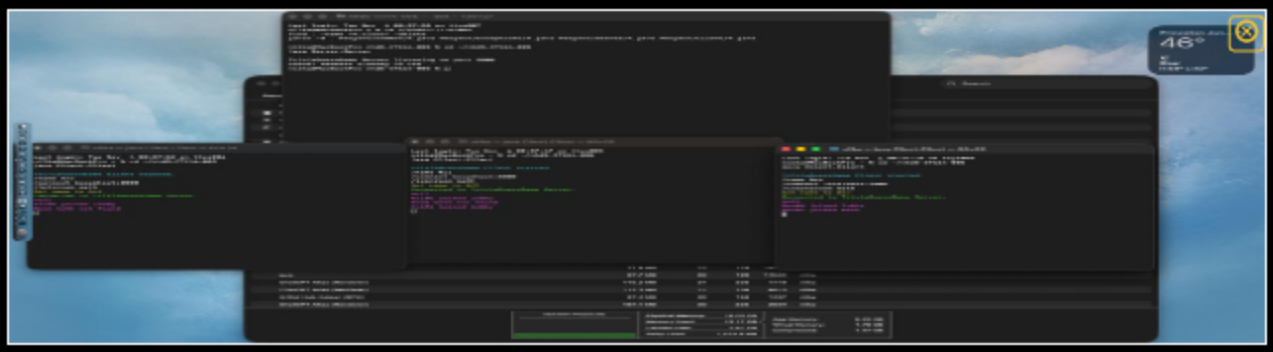
Progress: 100%

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



all four windows

Saved: 11/4/2025 12:41:47 AM

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 or /joinroom, the client program recognizes the command in its input loop and packages it into a Payload object. This payload specifies the requested action (create or join) and the room name, and it's sent to the server through the socket connection. On the server side, the ServerThread receives the payload and calls either handleCreateRoom() or handleJoinRoom() inside the Room class. These methods coordinate with the main Server instance to create new rooms, move the client into them, and broadcast updates to all connected users. As a result, clients can dynamically create and join rooms, with confirmation messages printed on both the client and server terminals.

Saved: 11/4/2025 12:41:47 AM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

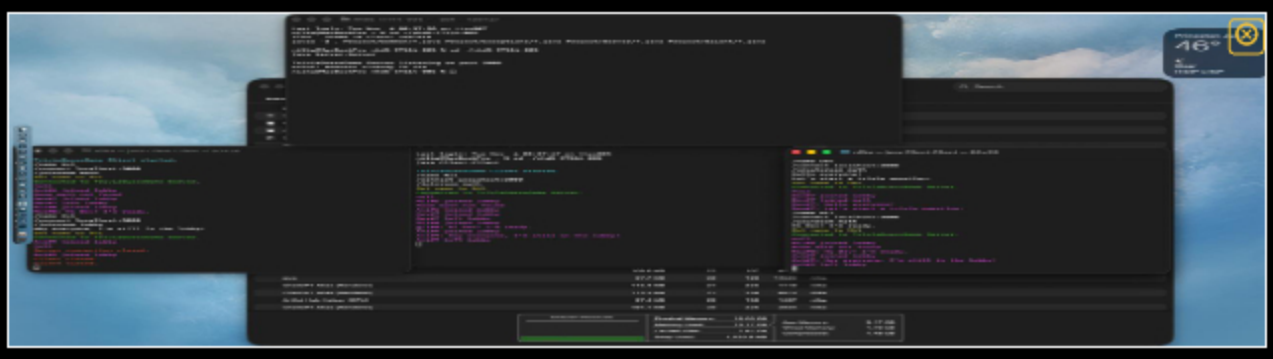
Progress: 100%

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



4 windows

4 Windows

client

serverthread

room

Saved: 11/4/2025 12:45:19 AM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

When a client types a message that isn't a command (it doesn't start with "/"), the client program packages it into a Payload object marked with the type MESSAGE and sends it to the server using the socket output stream. The server receives this payload inside the ServerThread and calls handleMessage() in the current Room object. The room then broadcasts the message to every connected client in that same room using the sendMessage() method. Each client listening to the

server prints the incoming messages in real time, allowing users in the same room to communicate instantly.

Saved: 11/4/2025 12:45:19 AM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

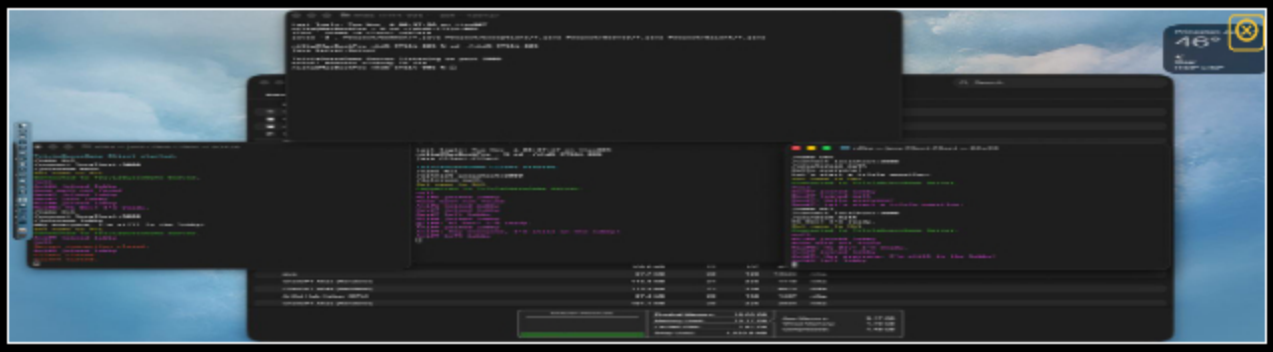
Progress: 100%

Part 1:

Progress: 100%

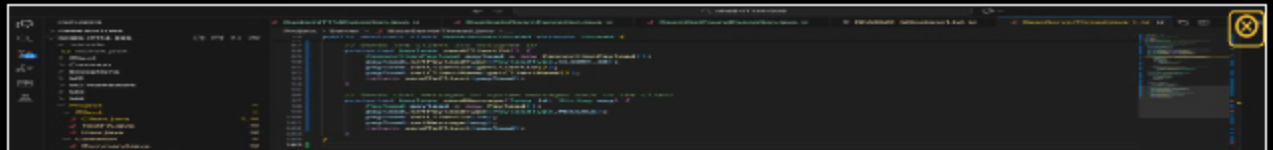
Details:

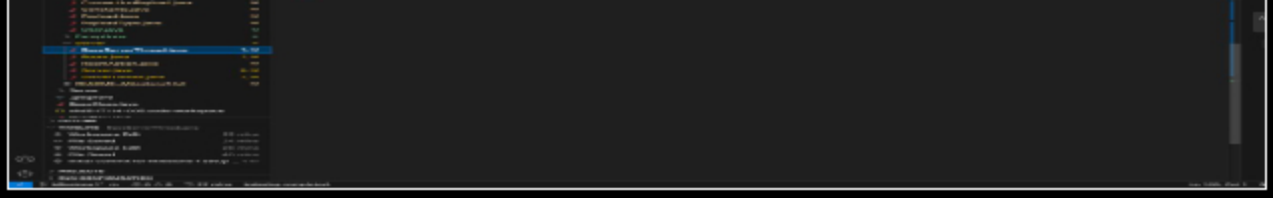
- Show examples of clients disconnecting (server should still be active)
- Show examples of server disconnecting (clients should be active but disconnected)
- Show examples of clients reconnecting when a server is brought back online
- Examples should include relevant messages of the actions occuring
- 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



4 windows

Client.java





BaseServerThread.java

Saved: 11/4/2025 12:49:20 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

When a client disconnects (either by typing /quit or losing connection), the client's socket is closed and the server receives a disconnect signal through the BaseServerThread. The server then removes that client from its current room using handleDisconnect() and broadcasts a message to other users in that room. If the server itself stops, all clients detect the broken connection and display "Server connection closed." When the server restarts, clients can reconnect using /connect, rejoining the lobby or their chosen room.

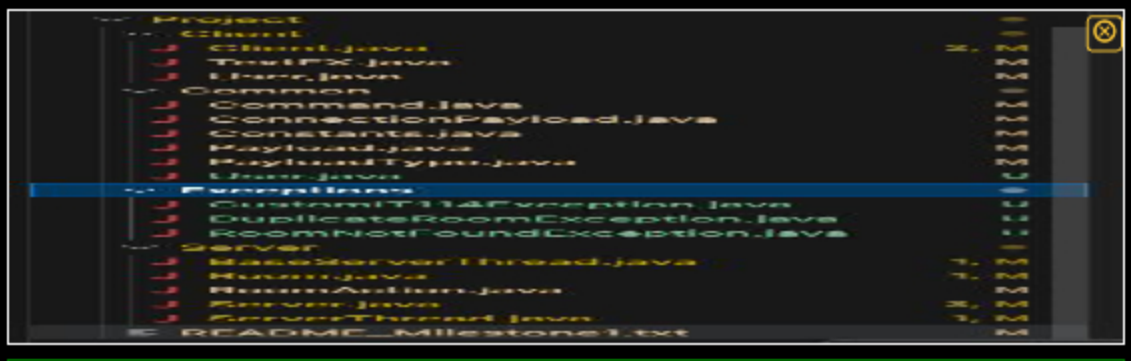
Saved: 11/4/2025 12:49:20 AM

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%



project setup

Saved: 11/4/2025 12:50:46 AM

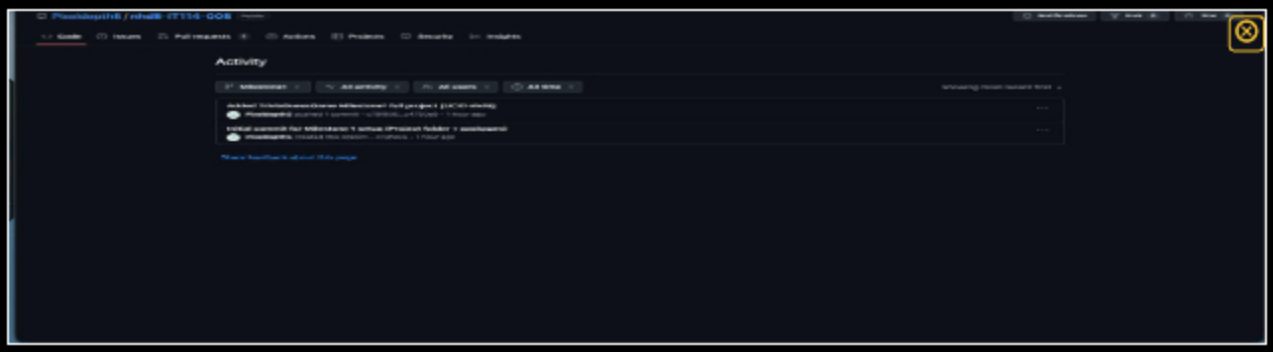
Task #2 (0.25 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commit history

Saved: 11/4/2025 12:53:18 AM

Part 2:

Progress: 100%

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

URL #1
https://github.com/Pixeldepth5/nhd5-IT114-005/pull/1?ref=Milestone1

URL
https://github.com/Pixeldepth5/nhd5-IT114-005/pull/1?ref=Milestone1

Saved: 11/4/2025 12:53:18 AM

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

dashboard

Saved: 11/4/2025 12:54:49 AM

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:

In this milestone, I learned how client-server programs actually use sockets and data streams to communicate. I had not understood how several users could connect to the same server at the same time, but now I see how threads make it possible. I also learned how to structure a Java project properly with different packages for organization and how to test commands like /connect, /createroom, and /joinroom through the terminal. In the end, this helped me understand the basic idea of networking logic behind multiplayer games or chat apps.

Saved: 11/4/2025 12:55:43 AM

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 the assignment was getting a basic project structure set up and a server up and running. Once I understood how to structure the folders into Client, Server, Common, and Exceptions packages, everything kind of clicked. The simple commands, like /name and /connect, were also fairly easy to implement as they had a pretty logical pattern behind them. Once the clients were able to actually connect to the server, everything started to feel more doable.

Saved: 11/4/2025 12:56:47 AM

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 most difficult part of this assignment was having multiple clients connect and speak properly with each other simultaneously. I encountered several errors while handling threads and making sure each client had its own connection without crashing the server. Understanding how the server handled rooms and synchronized methods took some time to understand, especially when testing different commands such as /createroom and /joinroom. It was quite confusing in the beginning, but once it clicked and I saw how each component worked with others, it came together.

Saved: 11/4/2025 12:57:43 AM