Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Milestone 1
Student: Graham B. (gb373)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 6/30/2025 3:33:44 PM
Updated: 6/30/2025 7:33:51 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/grading/gb373
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/view/gb373

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

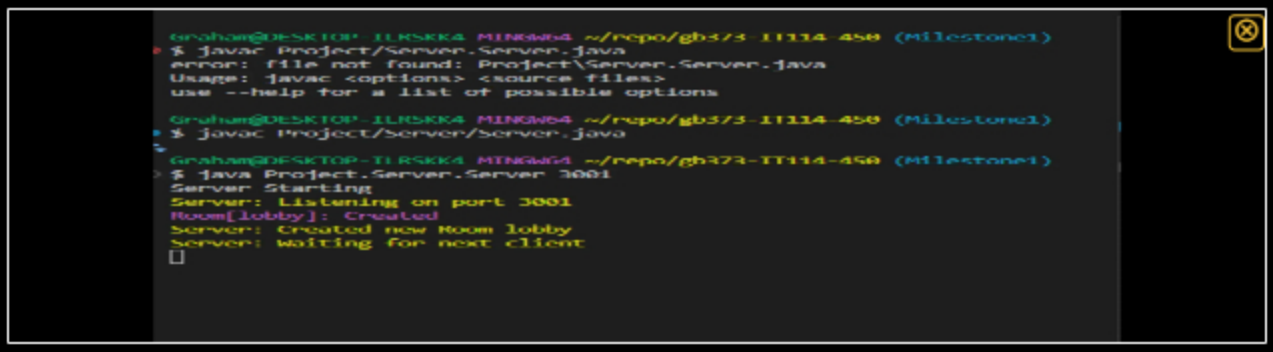
Progress: 100%

Part 1:

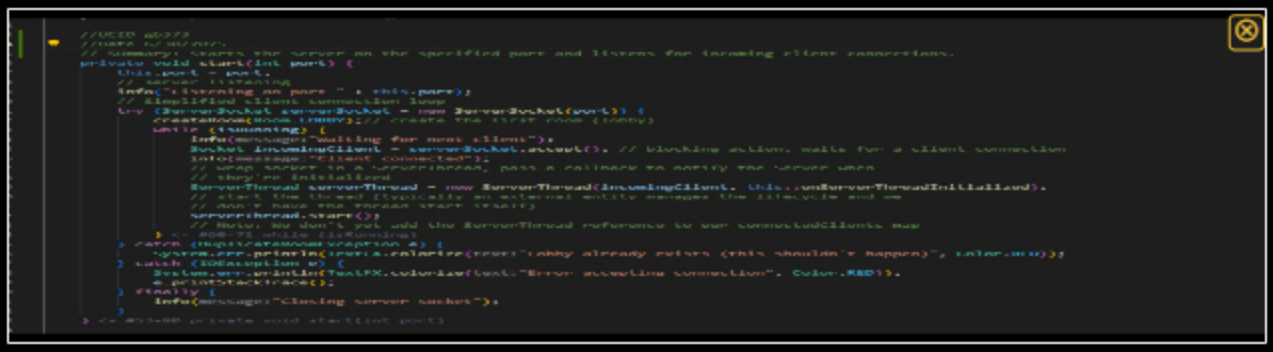
Progress: 100%

Details:

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



Output



Code referencing Section 1

Saved: 6/30/2025 3:36:28 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server uses a socket to connect to port 3001. It then uses a while loop to continuously check for client connections. Then, when the client connects, a new thread is created and puts the client into the default lobby.

Saved: 6/30/2025 3:36:28 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

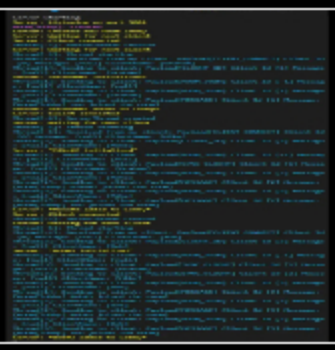
Progress: 100%

Part 1:

Progress: 100%

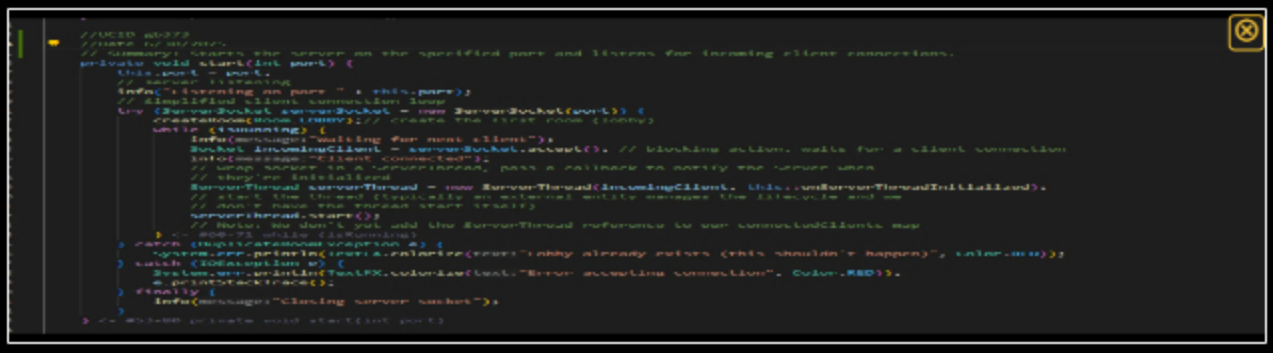
Details:

- 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



Server Output for section 2

3 clients connected to server



Code for section 2 showing is running

Code for setting client ids

Saved: 6/30/2025 3:49:53 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The server uses the is running to make connections for the client and loops it for each client. This means that the server creates a new thread for each client. With this, the server can run multiple clients at once with different threads.

Saved: 6/30/2025 3:49:53 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

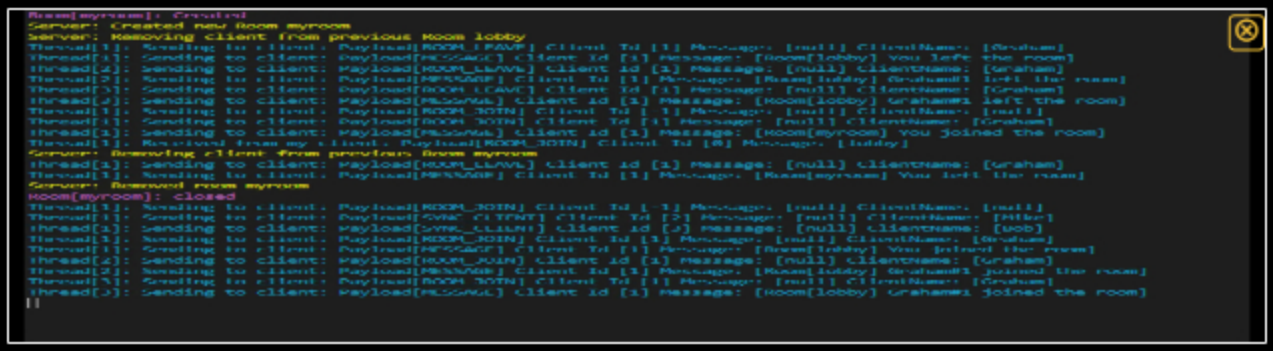
Progress: 100%

Part 1:

Progress: 100%

Details:

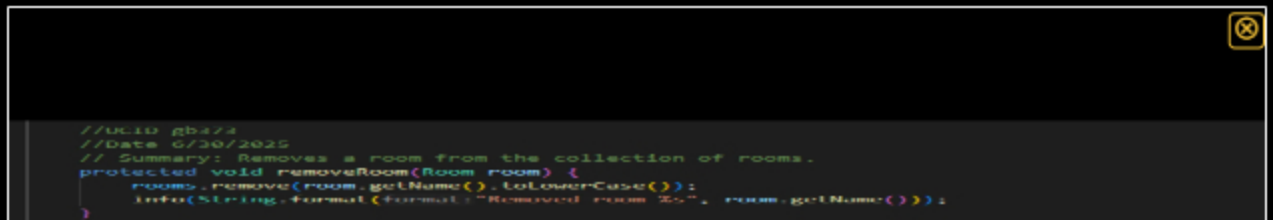
- Show the terminal output of rooms being created, joined, and removed (server-side)
- Show the relevant snippets of code that handle room management (create, join, leave, remove) (server-side)



Server side of room creation, joining, and removal.

create room code

join/leave room code



Remove Room Code

Saved: 6/30/2025 4:59:45 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

First, the default lobby is called lobby, and this is made when the server thread is initialized. Using createRoom, it creates a new room and checks if the room already exists. Clients can then use joinroom to join the room that has been created. This method handles the joining of rooms and the leaving of rooms by removing them from the previous one and removing them when the leave room is typed. When removeroom is used, this is typically when a room is empty and no client is in it.

Saved: 6/30/2025 4:59:45 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

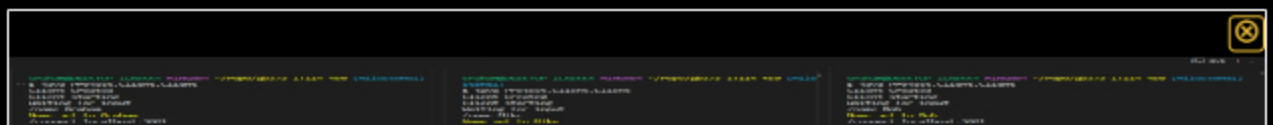
Progress: 100%

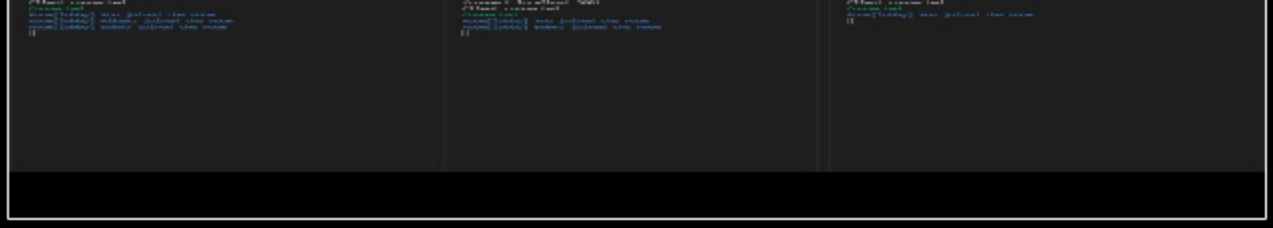
Part 1:

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 Connect and Name Output and Successful Connection

Code for /name in Client

code for /connect in client

Code for conformation

Saved: 6/30/2025 4:55:41 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 a client uses /name, it will get rid of the / and use the name that is given, and set it as the name of the client. For the /connect, the client would connect to the local host and find the port that the server is running on, which in this case is 3001. It then opens a socket for connection 5to the server, and if it is a success, the client will be notified that it is successful. The confirmation part works like it gets the client ID ready, then processes the client data then prints in green that they are connected to the server.

Saved: 6/30/2025 4:55:41 PM

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

Progress: 100%

Task #1 (2 pts.) - Evidence

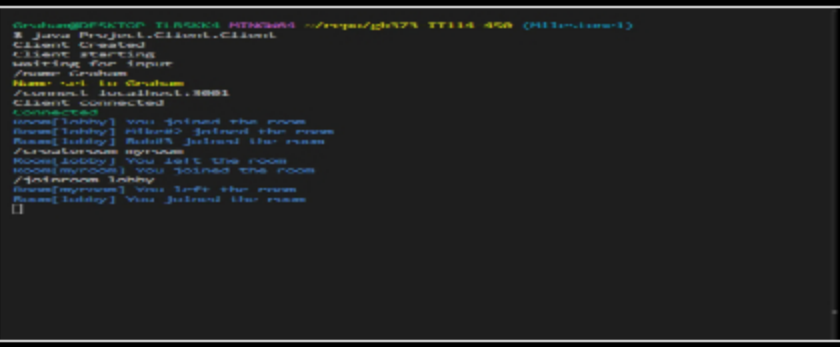
Progress: 100%

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



output for client creating room and joining room and sucess for both scenarios

code for both commands

code for sending any room actions to server

code for sending payloads to server

Saved: 6/30/2025 5:30:06 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 /createroom and /joinroom work with the client command where the client puts it in, then it sends the payload type to the corresponding command, where it will send it through the sendToServer method. Which will work if the client is connected. If the server receives the payload, it will send the messages back and create the room or have the client join the room.

Saved: 6/30/2025 5:30:06 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

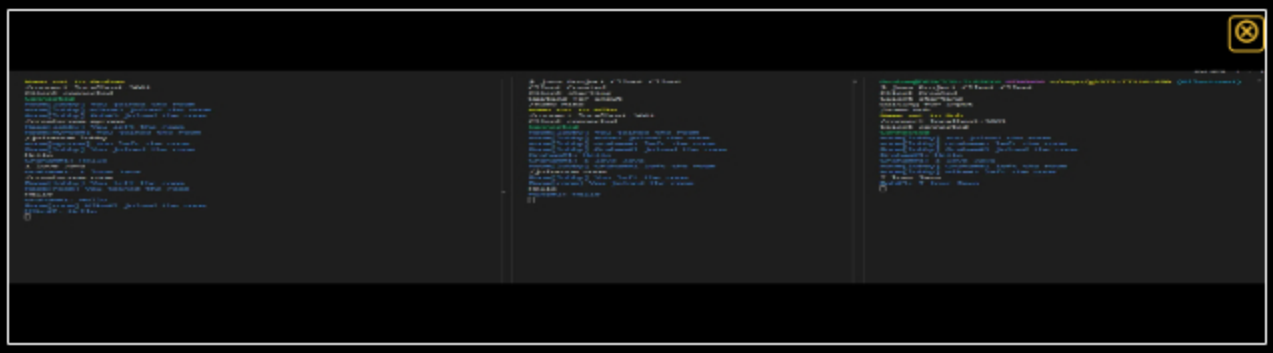
Progress: 100%

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



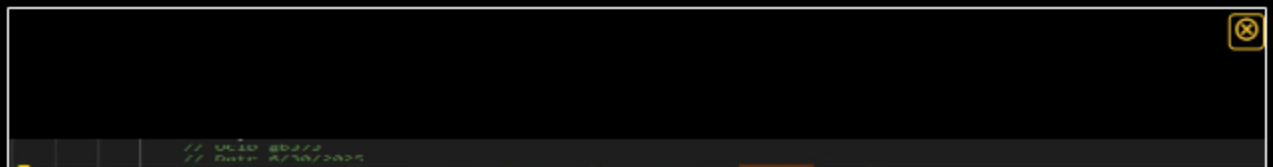
Messages from clients and also some that are in different rooms

```
//UCID gb373          You, 19 seconds ago • Uncommitted changes
//Date 6/4#/2025
// Summary: Sends a message to the server, which will then broadcast it to other
// clients in the room.
private void sendMessage(String message) throws IOException {
    Payload payload = new Payload();
    payload.setMessage(message);
    payload.setPayloadType(PayloadType.MESSAGE);
    sendToServer(payload);
} <- #ucid-gb7 private void sendMessage(String message) throws IOException
```

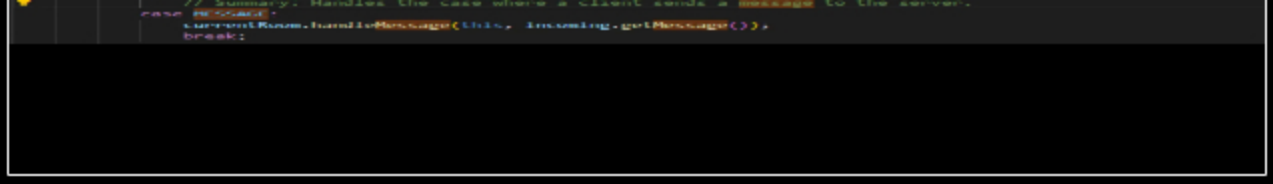
code for client side sending messages

```
            //ucid gb373
            //Date 6/30/2025
            // Summary: handles the payload types for messages between the client and server.
        case MESSAGE:
            processMessage(payload);
            break;
```

payload code for messages



```
// UCID gb373
// Date 6/30/2025
```



code for the case of message server side

code for sending message to all clients in a certain room.

Saved: 6/30/2025 5:58:26 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

When a client enters a message, it calls a send message payload with the type Message. On the server side, the server thread processes the payload and then identifies it as a message. It will then call the handleMessage method in which room that the client is currently in. For the room code, it uses a relay to send the message to all the clients in the same room as the sender.

Saved: 6/30/2025 5:58:26 PM

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

Progress: 100%

Task #1 (1 pt.) - Evidence

Progress: 100%

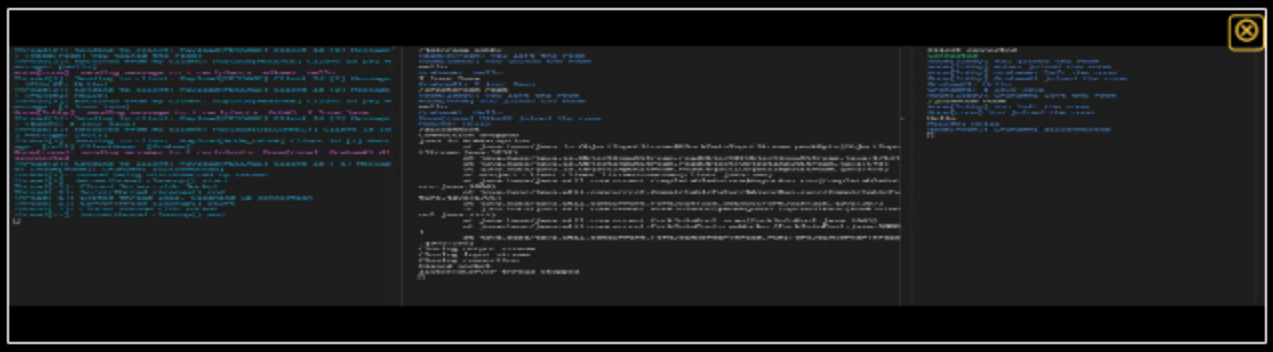
Part 1:

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 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



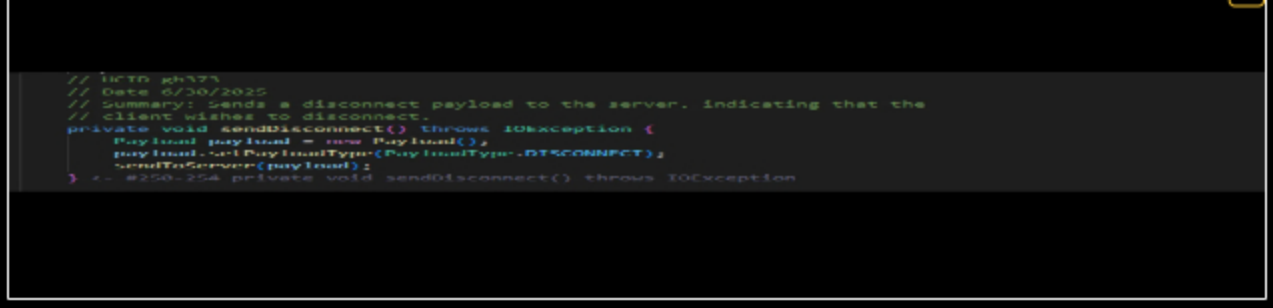
output for a client disconnecting

output for server disconnecting

output of reconnection after server comes back online

```
        // UCID gb373
        // Date: 6/10/2025
        // Summary: handles the /disconnect command to disconnect from the server
    } else if (Command.DISCONNECT.command.equalsIgnoreCase(text)) {
        sendDisconnect();
        waitCommand = true;
```

code for command on client side for disconnect



```
// UCID: gh377
// Date: 6/30/2025
// Summary: Sends a disconnect payload to the server, indicating that the
// client wishes to disconnect.
private void sendDisconnect() throws IOException {
    Payload payload = new Payload();
    payload.setPayloadType(PayloadType.DISCONNECT);
    sendToServer(payload);
} <- #250-254 private void sendDisconnect() throws IOException
```

sends disconnect payload to server from client side

```
// UCID: gh377
// Date: 6/30/2025
// Summary: Listens for incoming messages from the server and handles disconnections
private void listenToServer() {
    try {
        while (isRunning && isConnected()) {
            Payload fromServer = (Payload) in.readObject(); // blocking read
            if (fromServer != null) {
                processPayload(fromServer);

            } else {
                System.out.println("Server disconnected");
                break;
            }
        } <- #316-324 while (isRunning && isConnected())
    } catch (ClassCastException | ClassNotFoundException exc) {
        System.err.println("Error reading object as expected type: " + exc.getMessage());
        exc.printStackTrace();
    } catch (IOException e) {
        if (isRunning) {
            System.out.println("Connection dropped");
            e.printStackTrace();
        }
    } finally {
        closeEverything();
    }
    System.out.println("ListenToServer thread stopped");
} <- #313-327 private void listenToServer()
```

code for messageas to do with disconnections

```
// UCID: gh377
// Date: 6/30/2025
// Summary: Closes the server connection and associated resources, ensuring a clean
// shutdown.
private void closeEverything() {
    try {
        if (out != null) {
            System.out.println("Closing output stream");
            out.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        if (in != null) {
            System.out.println("Closing input stream");
            in.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        if (server != null) {
            System.out.println("Closing connection");
            server.close();
            System.out.println("Closed socket");
        } <- #825-827 if (server != null)
    } catch (IOException e) {
        e.printStackTrace();
    }
} <- #808-831 private void closeEverything()
```

code for closing the client connection

```
//UCID: gh377
//Date: 6/30/2025
// Summary: Private constructor sets up shutdown hook for graceful JVM shutdown & message.
private Server() {
    Runtime.getRuntime().addShutdownHook(new Thread(() -> {
        Info("\n\nLogger: JVM is shutting down. Performing cleanup tasks...");
        shutdown();
    }));
} <- #55-57 private Server()
```

code on server side for shutdown hook for JVM shutdown

```
//UCID: gh377
//Date: 6/30/2025
// Summary: Shuts down the server by disconnecting all clients and removing rooms.
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();
    }
} <- #45-57 private void shutdown()
```

code for server side for when shutdown happens, it disconntects clients and removes rooms.

Saved: 6/30/2025 6:28:40 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

On the client side, the client uses /disconnect command to start disconnect. When this command is made, it closes client streams, socket connections, and any threads that are associated with it or listening. It will then send a disconnect payload to the server, and then message to the user that the connection has been dropped. On the server side, when the server disconnects, it registers the hook for the JVM shutdown. It then uses the shutdown method to close all rooms and disconnect every client, but the clients are still active. Messages are sent to confirm shutdown, and the server will be disconnected.

Saved: 6/30/2025 6:28:40 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%

proper workspace structure for my project folder

Saved: 6/30/2025 6:29:58 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

commits for milestone 1

Saved: 6/30/2025 6:32:09 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/GrahamBlack10/gb373-IT114-450/pull/6

URL
https://github.com/GrahamBlack1

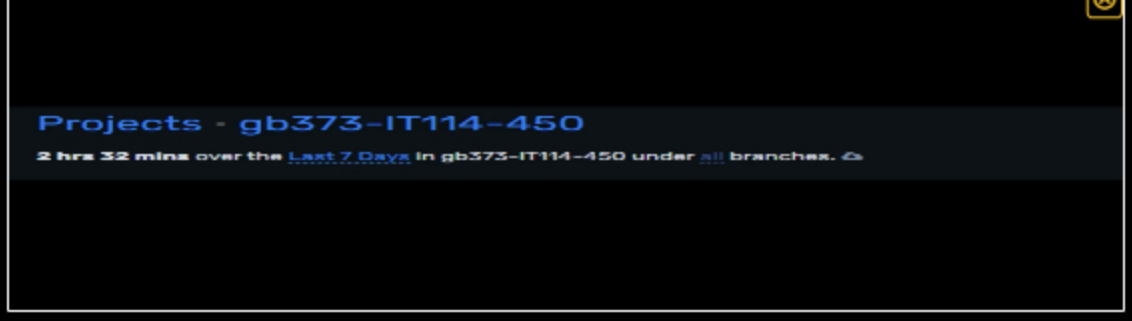
Saved: 6/30/2025 6:32:09 PM

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



overall time

individual time

Saved: 6/30/2025 6:37:08 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 learn more about sockets and the different features that they can do. I learned more about advanced features of the server, as well as how clients can create their rooms and the server can hold them. I found out about how clients can be created with their thread and their own name to have differences from other clients. I also learn about how servers can run multiple clients and how they can disconnect, but the clients will still be active.

Saved: 6/30/2025 7:26:39 PM

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

assignment?

Progress: 100%

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

Your Response:

The easiest part of this assignment was understanding the connection between client and server. What I mean by that is how they connected through a port and how the code did it. This was mainly because this was in the previous socket parts. I also found that the payloads and how they are created, and each type, were easy to understand, and the concept behind them was easy to understand, too.

Saved: 6/30/2025 7:31:08 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 the assignment was understanding exceptions and the differences between them. I found that I had trouble understanding why there was an exception when the client disconnected, and I had to learn more about it. Also, I had some trouble figuring out the process of sending messages. This included the process of sending messages between client and server, and how the room was put into effect, meaning where a sender would only send a message to clients in the same room.

Saved: 6/30/2025 7:33:51 PM