

# Submission Worksheet

**CLICK TO GRADE**

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-5-project-milestone-1/grade/mcp62>

Course: IT114-003-F2024

Assignment: [IT114] Module 5 Project Milestone 1

Student: Michael P. (mcp62)

## Submissions:

Submission Selection

1 Submission [submitted] 10/21/2024 11:09:11 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**

Group: Start Up

Tasks: 2

Points: 3

100%

▲ COLLAPSE ▲

**Task**

Group: Start Up

Task #1: Start Up

Weight: ~50%

Points: ~1.50

100%

▲ COLLAPSE ▲

**i 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**

Group: Start Up

Task #1: Start Up

100%

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

**Task Screenshots**

Gallery Style: 2 Columns

4      2      1

```
Listening for client connection...  
Client connected from 127.0.0.1:53000  
Client disconnected from 127.0.0.1:53000
```

server connecting and waiting for client

**Caption(s) (required) ✓**Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Start Up

Task #1: Start Up

100%

Sub Task #2: Show the Server Code that listens for connections

**Task Screenshots**

4 2 1

```

private void start(int port) {
    this.port = port;
    // server listening
    System.out.info("Listening on port " + this.port);
    try (ServerSocket serverSocket = new ServerSocket(port)) {
        createLobby(); // create the lobby room
        while (true) {
            logger.info("Waiting for client connection");
            Socket incomingClient = serverSocket.accept(); // blocking until one or a client connection
            ServerThread serverThread = new ServerThread(incomingClient);
            // wrap socket in server thread since it's unlikely we'll verify user server may be
            // initialized
            serverThread.start(); // each server thread entity manages the lifecycle and we
            // don't have the thread start itself
            serverThread.start();
            logger.info("Accepted connection from client");
        }
    } catch (IOException e) {
        logger.error("Error accepting connection", e);
    } finally {
        System.out.info("Exiting server");
    }
}

```

code that starts port

**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 `start(int port)` method sets up a server that listens for client connections on a specified port. It creates a `ServerSocket` and initializes a lobby room for clients. The method enters a loop to accept incoming connections, logging each one, and spawns a new `ServerThread` for each client to handle interactions concurrently. It includes error handling for connection issues and ensures proper resource cleanup upon shutdown.

**Sub-Task**

Group: Start Up

100%

Task #1: Start Up

Sub Task #3: Show the Client starting via command line

**Task Screenshots**

4 2 1

The image shows two terminal windows. The left terminal window displays client-side logs, including messages like "Connected to lobby", "Client created", and "Client joined lobby". The right terminal window displays server-side logs, including "Listening on port 8080", "Accepted connection from client", and "Client joined lobby". Both terminals show the process of connecting to a lobby and joining it.

shows two terminals on the left are the client

**Caption(s) (required)** ✓Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Start Up

100%

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

```
private boolean connect(String address, int port) {  
    try {  
        server = new Socket(address, port);  
        // message to send to server  
        ObjectOutputStream out = new ObjectOutputStream(server.getOutputStream());  
        // message to listen to server  
        in = new ObjectInputStream(server.getInputStream());  
        loggerUtil.DESTACK.info("Client connected");  
        // use compileTimeValue to run listenToServer() in a separate thread  
        // consider adding this to a separate thread  
        // consider adding this to a separate thread  
        new Thread(listenToServer).start();  
    } catch (IOException e) {  
        loggerUtil.DESTACK.error("Unknown host", e);  
    } catch (Exception e) {  
        loggerUtil.DESTACK.error("Message TO exception", e);  
    }  
    return isConnected;  
} <-- end private boolean connect(String address, int port)
```

code that checks connection

**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:

The connect(String address, int port) method establishes a connection to a server at the specified address and port. It creates input and output streams for communication and logs a success message upon connection. The method runs listenToServer() in a separate thread to handle incoming messages concurrently. It includes error handling for connection issues and returns the connection status.

End of Task 1

### Task

Group: Start Up

Task #2: Connecting

Weight: ~50%

Points: ~1.50

100%

**▲ 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

Group: Start Up

Task #2: Connecting

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

100%

## Task Screenshots

Gallery Style: 2 Columns

```
4 2 1
[client1] 127.0.0.1:52000-localhost-00000000000000000000000000000000
[client2] 192.168.1.10:52000-192.168.1.10-00000000000000000000000000000000
[client3] 192.168.1.11:52000-192.168.1.11-00000000000000000000000000000000
[client4] 192.168.1.12:52000-192.168.1.12-00000000000000000000000000000000
[server] 192.168.1.13:8080-localhost-00000000000000000000000000000000
```

Shows 3rd client on far right

### 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
$ ./client -connect 127.0.0.1:8080
Connected to 127.0.0.1:8080
```

shows process commands for name

```
private final String SERVER_IP = null;
private final String SERVER_PORT = null;
private final String LOCALHOST_IP = null;
final Pattern LOCALHOST_PATTERN = null;
final String[] compile(String... args) {
    final Pattern[] patterns = new Pattern[args.length];
    for (int i = 0; i < args.length; i++) {
        patterns[i] = Pattern.compile(args[i]);
    }
    return patterns;
}
```

shows regex pattern for local host connectivity

### 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:

initializes a logging utility for the client, configuring it to limit log file size to 2MB, keep one file, and store logs in "client.log." It declares several member variables: a Socket for the server connection, output and input streams for communication, and patterns to match connection commands for both IP addresses and localhost. Additionally creates a regex pattern to verify localhost connect command. Proccess commands goes over /name functionality

## End of Group: Start Up

Task Status: 2/2

### Group



Group: Communication

Tasks: 2

Points: 3

[^ COLLAPSE ^](#)

### Task



Group: Communication

Task #1: Communication

Weight: ~50%

Points: ~1.50

[^ COLLAPSE ^](#)

### i 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



Group: Communication

Task #1: Communication

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

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

Shows clients sending and receiving

### Caption(s) (required) ✓

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

### Sub-Task

Group: Communication

100%

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

```

    /**
     * protected boolean sendPayload(payload) {
     *     if (!running)
     *         return true;
     *     try {
     *         out.writeObject(payload);
     *         out.flush();
     *         return true;
     *     } catch (IOException e) {
     *         logMessage("Error sending message to client (most likely disconnected)");
     *         // If this fails, we'll re-open the socket
     *         // If we're still here, cleanup()
     *         cleanup();
     *     }
     *     return false;
     * }
     */
    protected boolean sendPayload(payload) {
        if (!running)
            return true;
        try {
            out.writeObject(payload);
            out.flush();
            return true;
        } catch (IOException e) {
            logMessage("Error sending message to client (most likely disconnected)");
            // If this fails, we'll re-open the socket
            // If we're still here, cleanup()
            cleanup();
        }
        return false;
    }

```

send over socket w payload

**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:

send(Payload payload) method attempts to send a Payload object to the client using the output stream (out). If the client is not running, it returns true immediately. Upon successful writing and flushing of the object, it returns true. If an IOException occurs (indicating a potential disconnection), it logs an error message, calls a cleanup method to handle the disconnection, and returns false.

Sub-Task

Group: Communication

100%

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

```

    /**
     * Sends a message with the author/source identifier
     *
     * @param senderId
     * @param message
     * @return true (@link send(payload))
     */
    public boolean sendMessage(long senderId, String message) {
        Payload p = new Payload();
        p.setClientID(senderId);
        p.setMessage(message);
        p.setPayloadType(PayloadType.MISSIVE);
        return send(p);
    }
    < #204-210 public boolean sendMessage(long senderId, String message)
    ...

```

server side relay

**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 sendMessage(long senderId, String message) method constructs a Payload object to represent a message being sent by a client. It sets the client's ID, the message content, and specifies the payload type as a message. Finally, it calls the send(p) method to transmit the payload, facilitating the communication from the client to the server or other clients.

Sub-Task

Group: Communication

100%

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

```
case PayloadType.CLIENT_ID: // get id assigned
    ConnectionPayload cp = (ConnectionPayload) payload;
    processClientId(cp.getClientId(), cp.getClienteName());
    break;
case PayloadType.SYNC_CLIENT: // silent add
    cp = (ConnectionPayload) payload;
    processClientIdName(cp.getClientId(), cp.getClienteName());
    break;
case PayloadType.DISCONNECT: // remove a disconnected client (mostly for the specific message vs leaving
    cp = (ConnectionPayload) payload;
    processDisconnect(cp.getClientId(), cp.getClienteName());
    break;
case PayloadType.ROOM_LIST: // room list
    ConnectionPayload cp = (ConnectionPayload) payload;
    processRoomAction(cp.getClientId(), cp.getClienteName(), cp.getMessage(), cp.isConnect());
    break;
case PayloadType.READY: // displays a received message
    processMessage(payload.getClientId(), payload.getMessage());
    break;
case PayloadType.READY:
    ReadyPayload rp = (ReadyPayload) payload;
    processReadyStatus(rp.getClientId(), rp.isReady(), quiet);
    break;
case PayloadType.GAME_READY:
```

displays message switch case

**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 processPayload(Payload payload) method handles different types of incoming payloads by logging the received data and processing it based on its type using a switch statement. Each case triggers specific actions, such as updating client information, managing room actions, displaying messages, or changing readiness statuses, with dedicated methods handling the actual logic for each type.

End of Task 1

Task

Group: Communication

100%

Task #2: Rooms

Weight: ~50%

Points: ~1.50

**i 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**

Group: Communication

100%

Task #2: Rooms

Sub Task #1: Show Clients can Create Rooms

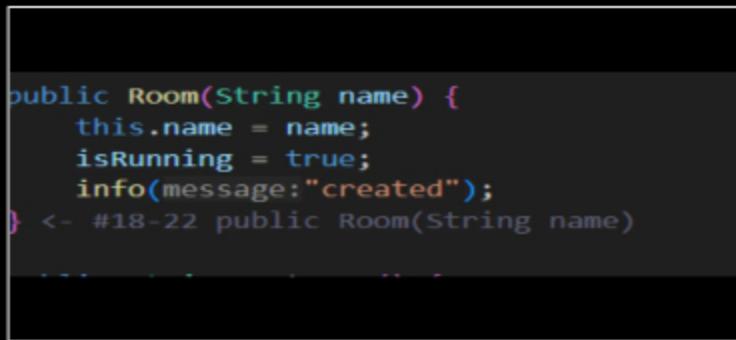
**▣ Task Screenshots**

Gallery Style: 2 Columns

4

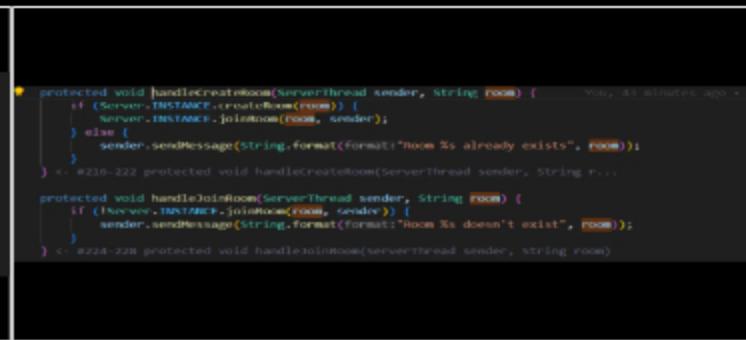
2

1



```

public Room(String name) {
    this.name = name;
    isRunning = true;
    info(message:"created");
}
<= #18-22 public Room(String name)
  
```



```

protected void handleCreationOn(ServerThread sender, String room) {
    if (Server.INSTANCE.createRoom(room)) {
        Server.INSTANCE.joinRoom(room, sender);
    } else {
        sender.sendMessage(String.format(format:"Room %s already exists", room));
    }
} <= #216-222 protected void handleJoinRoom(ServerThread sender, String room) {
    if (!Server.INSTANCE.joinRoom(room, sender)) {
        sender.sendMessage(String.format(format:"Room %s doesn't exist", room));
    }
} <= #224-228 protected void handleJoinRoom(ServerThread sender, String room)
  
```

room constructor

cases for join and create room

**Caption(s) (required)** ✓Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Communication

100%

Task #2: Rooms

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

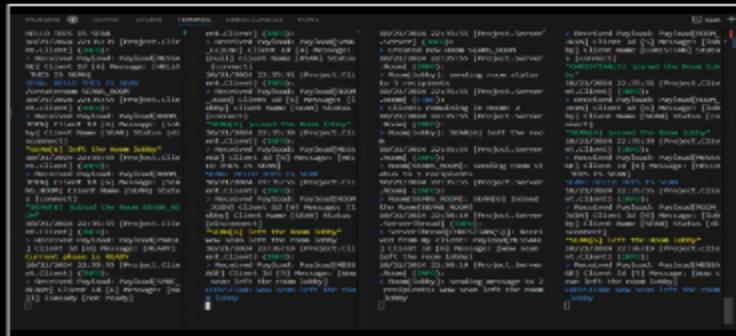
**▣ Task Screenshots**

Gallery Style: 2 Columns

4

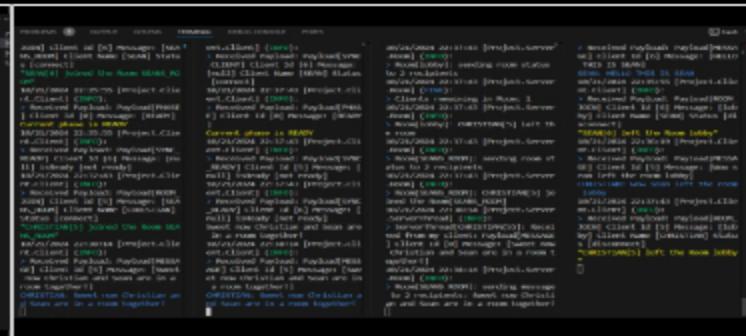
2

1



```

16/03/2004 22:01:10 [Project,Client] > received payload: [Project,Client]
16/03/2004 22:01:10 [Project,Client] > joined Client id [6] messages sent
16/03/2004 22:01:10 [Project,Client] > Client [6] joined the room [sean]
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [6] sending room status to 2 recipients
  
```



```

16/03/2004 22:01:10 [Project,Client] > joined Client id [5] messages sent
16/03/2004 22:01:10 [Project,Client] > Client [5] joined the room [sean]
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
16/03/2004 22:01:10 [Project,Client] > Client [5] sending room status to 2 recipients
  
```

Shows SEAN leaving room and not getting message in lobby

Shows christian joining seans room and ryan not getting

**Caption(s) (required)** ✓Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Communication

Task #2: Rooms

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

100%

**Task Screenshots**

Gallery Style: 2 Columns

4 2 1

```

private void sendCreateRoom(String room) {
    payload p = new Payload();
    p.setPayloadType(Payloadtype.ROOM_CREATE);
    p.setMesage(room);
    send(p);
} <- static private void sendCreateRoom(String room)

/** 
 * needs the room name we intend to join|      view, 33 minutes ago
 * @param room
 */
private void sendJoinRoom(String room) {
    payload p = new Payload();
    p.setPayloadType(Payloadtype.ROOM_JOIN);
    p.setMesage(room);
    send(p);
} <- static private void sendJoinRoom(String room)

/** 
 * tells the server-side we want to disconnect
 */
private void sendDisconnect() {
    payload p = new Payload();
    p.setPayloadType(Payloadtype.DISCONNECT);
    send(p);
} <- static private void sendDisconnect()

```

Client side create, join and leave methods

**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:

`sendCreateRoom(String room):` Constructs a Payload indicating a request to create a new room with the specified name and sends it.

**Sub-Task**

Group: Communication

Task #2: Rooms

Sub Task #4: Show the ServerThread/Room code handling the create/join process

100%

**Task Screenshots**

Gallery Style: 2 Columns

4 2 1

```

protected void processPayload(payload payload) {
    try {
        switch (payload.getPayloadType()) {
            case CLIENT_CONNECT:
                connectionpayload cp = (connectionpayload) payload;
                setClientName(cp.getClientName());
                break;
            case MESSAGE:
                currentroom.sendMessage(this, payload.getMessage());
                break;
            case ROOM_CREATE:
                currentroom.handleCreateRoom(this, payload.getMessage());
                break;
            case ROOM_JOIN:
                currentroom.handleJoinRoom(this, payload.getMessage());
                break;
            case ROOM_LIST:
                currentroom.handleListRooms(this, payload.getMessage());
                break;
            case DISCONNECT:
                currentroom.disconnect(this);
                break;
            case UNKNOWN:
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

protected synchronized void addClient(ServerThread client) {
    IP (blocking) { // block until IP from client is known
        return;
    }
    if (currentroom.containsClient(client.getNickname())) {
        errorMessage("Attempting to add a client that already exists in the room");
        return;
    }
    client.setNickname(client.getNickname());
    client.setServerThread(this);
    client.setRoom(currentroom);
    currentroom.getClients().add(client);
    currentroom.setClientCount(currentroom.getClientCount() + 1);
    synchronized (clients) {
        clients.add(client);
    }
    discovering.currentServer[0].join(this.nickname, client.getNickname(), client.getNickname(), nickname);
}

```

**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:

`addClient(ServerThread client)` method adds a new client to a room, ensuring thread safety with the synchronized keyword. It first checks if the room is running; if not, it exits without action. If the client is already in the room, it logs a message and returns. If the client is new, it adds them to the clientsInRoom map

**Sub-Task**

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

```
protected boolean joinRoom(String name, ServerThread client) {
    final String nameCheck = name.toLowerCase();
    if (rooms.containsKey(nameCheck)) {
        return false;
    }
    Room current = client.getCurrentRoom();
    if (current != null) {
        current.removeClient(client);
    }
    Room next = rooms.get(nameCheck);
    next.addClient(client);
    return true;
} <-- 810-142 protected boolean joinRoom(String name, ServerThread client)
```

server code for join

**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:

allows a client to join a specified room. It first converts the room name to lowercase for consistency and checks if the room exists in the rooms map. If the room does not exist, it returns false. If the client is already in another room, it removes them from that room by calling removedClient(client). It then retrieves the specified room and adds the client to it by calling addClient(client), returning true to indicate a successful join.

**Sub-Task**

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**

4 2 1

```

Client 1: [Client 1] > hi
Client 2: [Client 2] > hello
RYAN: [RYAN] > hi

```

shows (far right) RYAN user is isolated from far left 2 clients

### Caption(s) (required) ✓

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

## ≡ Task Response Prompt

*Briefly explain why/how it works this way*

Response:

This is done via server threads, rooms manage said threads and once a new room is made it sync only those on the same thread.

End of Task 2

End of Group: Communication

Task Status: 2/2

Group

Group: Disconnecting/Termination

Tasks: 1

Points: 3

▲ COLLAPSE ▲

Task

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)



**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

```
PROBLEMS DASHBOAR OPENED TERMINAL EDITOR CONSOLE PORTS
d:\base\src> cd \base\src\branches\j2se-14\j2se\src\java\lang\java.net
d:\base\src> javac -cp . -d lib Main.java
d:\base\src> java -cp lib Main
[Client] (2023) 2023-07-17 [Project.Client]
[Client] [INFO] >>> Server disconnected
[Client] [INFO] >>> Server disconnected.
```

shows gracefull dc

**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**

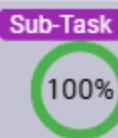
4      2      1

```
public void close() {
    // attempt to gracefully close and migrate clients
    if (!clientsInRoom.isEmpty()) {
        String msg = String.format("Room is shutting down, migrating to lobby");
        infoString = String.format("Reporting %s clients", name, clientsInRoom.size());
        clientsInRoom.values().removeIf(client -> {
            Server.getInstance().joinRoom(name, client);
            return true;
        });
    }
    <-- while loop if (!clientsInRoom.isEmpty())
    RoomManager roomManager = RoomManager.this;
    roomManager.setLobby(true);
    clientsInRoom.clear();
    infoString = String.format("Forced %s leave", name);
}
<- #110-120 public void close()
// send sync data to client(s)
```

graceful dc

**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:**

closes a chat room by notifying clients of the shutdown, migrating them to a lobby, and then removing the room from the server while clearing its client list.



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

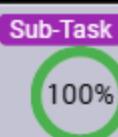
4 2 1

The screenshot displays two terminal windows. The left window shows client activity: a message from 'Client 1' to 'Client 2' at 22:59:39, followed by a message from 'Client 1' to 'Client 3' at 22:59:39. The right window shows the server log, which includes a message from 'Client 1' at 22:59:39 indicating it has moved to a lobby. The log also shows other clients ('Client 2' and 'Client 3') and their respective messages.

shows other clients sending a message and server detects it after D/c

**Caption(s) (required) ✓**

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



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

The screenshot shows a Java code snippet for room removal. It defines a method named 'removeRoom' that takes a room object as a parameter. Inside the method, it removes the room from a list of rooms and prints a message to the console indicating the room has been removed. The code uses the 'logger' utility class to log the message.

room removing

**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:

removes a specified room from the server's list of rooms, logging the action for informational purposes. The room name is converted to lowercase to ensure case-insensitive removal.

End of Task 1

End of Group: Disconnecting/Termination

Task Status: 1/1

Group

Group: Misc

Tasks: 3

Points: 1

100%

▲ COLLAPSE ▲

Task

Group: Misc

Task #1: Add the pull request link for this branch

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲

## 🔗 Task URLs

URL #1

<https://github.com/Onervv/mcp62-IT114-003/pull/12>

URL

<https://github.com/Onervv/mcp62-IT114-003/pul>

End of Task 1

Task

Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

100%

▲ COLLAPSE ▲

### ⓘ Details:

Few related sentences about the Project/sockets topics



## ≡ Task Response Prompt

Response:

Response:

Since I did a previous pull to milestone 1 most of the files were interchanged between milestone1 and 2 greatly increasing the difficulty of finding the related code however i tried my best to find the declarations where it was mostly handling said component.

End of Task 2

Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

**i** Details:

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

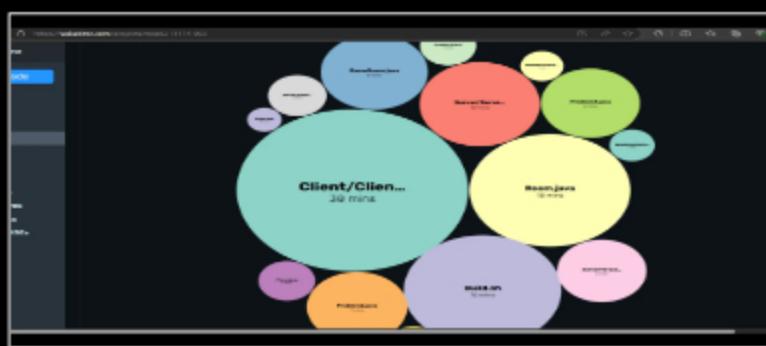
The duration isn't considered for grading, but there should be some time involved.



## Task Screenshots

Gallery Style: 2 Columns

4                    2                    1



waka

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment