Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-5-project-milestone-1/grade/rra23

Course: IT114-003-F2024
Assigment: [IT114] Module 5 Project Milestone 1
Student: Rahid A. (rra23)

Submissions:

Submission Selection

1 Submission [submitted] 10/17/2024 1:13:34 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

100%

Group: Start Up
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Start Up
Task #1: Start Up
Weight: ~50%
Points: ~1.50

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

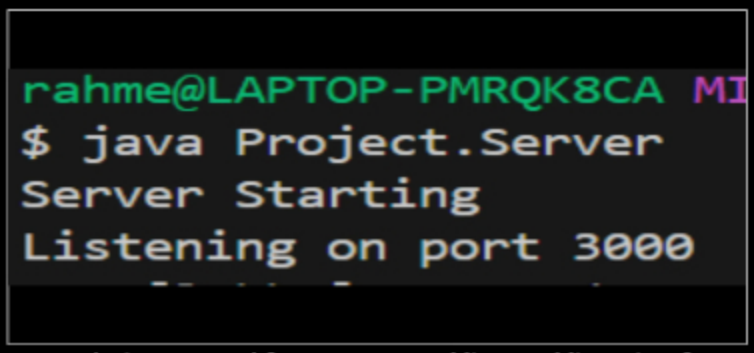
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server being started from command line and listening for connections

Caption(s) (required) ✓

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

Sub-Task

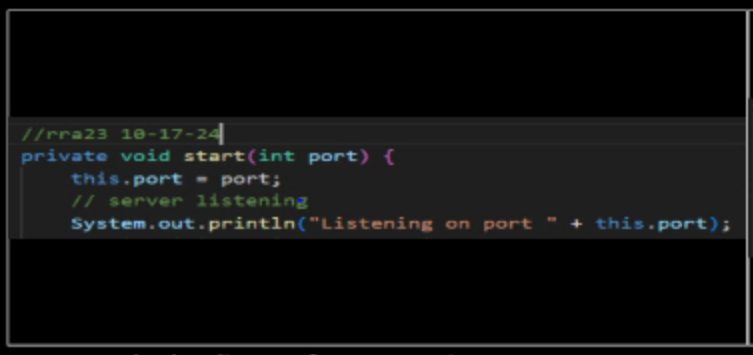
100%

Group: Start Up
Task #1: Start Up
Sub Task #2: Show the Server Code that listens for connections

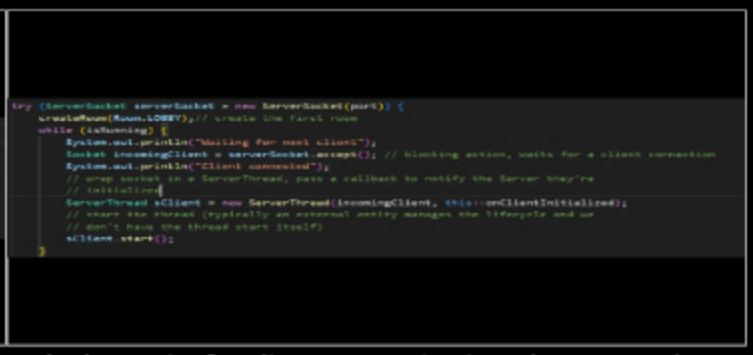
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server code that listens for connections



code that waits for client connection in order to start the serverthread

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:

In the code that is being shown, the server is setting the port to the port that has been passed through (in this example, it is port 3000) and then it is listening for any active connections that come through the port. If a connection comes through, it starts the ServerThread

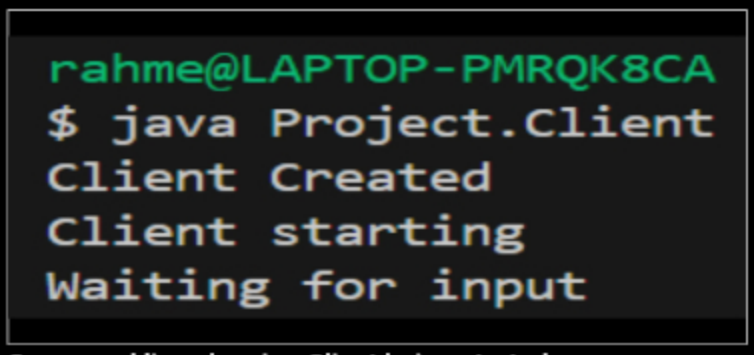
Sub-Task 100%

Group: Start Up
Task #1: Start Up
Sub Task #3: Show the Client starting via command line

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Command line showing Client being started

Caption(s) (required) ✓

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

Sub-Task 100%

Group: Start Up
Task #1: Start Up

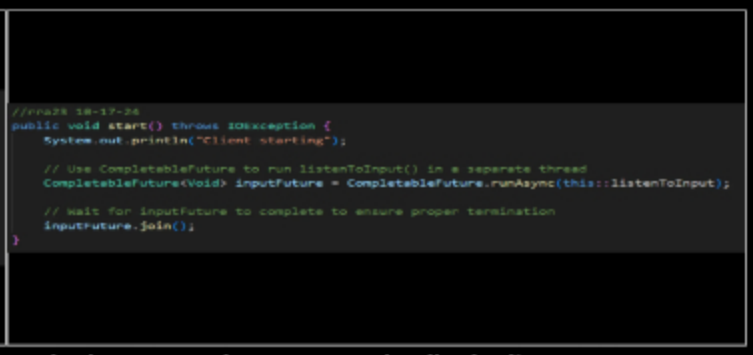
Sub Task #4: Show the Client Code that prepares the client and waits for user input

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client code that waits for user input



Code that starts the server and calls the listenToInput method

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 code uses try with resources to create a Scanner object called si. This object is then used in a while loop to see if while the program is running, is the user typing anything into the command line. Then it takes whatever the user typed and stores it in a variable called line. If the message matches a Client Command, it sends the message to the server with the command, if it does not match a Client Command, it prints a message to the screen.

End of Task 1

Task

100%

Group: Start Up
Task #2: Connecting
Weight: ~50%
Points: ~1.50

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

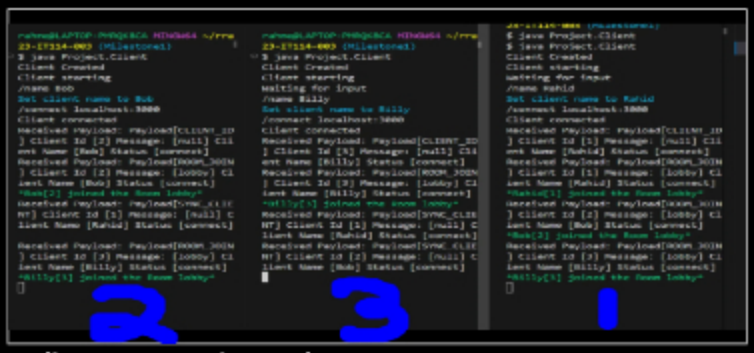
100%

Group: Start Up
Task #2: Connecting
Sub Task #1: Show 3 Clients connecting to the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1



3 clients connecting to the server

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

```
//rra23 10-17-24       You, 1 second ago • Uncommitted changes
private boolean connect(String address, int port) {
    try {
        server = new Socket(address, port);
        // channel to send to server
        out = new ObjectOutputStream(server.getOutputStream());
        // channel to listen to server
        in = new ObjectInputStream(server.getInputStream());
        System.out.println("Client connected");
        // Use CompletableFuture to run listenToServer() in a separate thread
        CompletableFuture.runAsync(this::listenToServer);
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return isConnected();
}
```

Client code that connects it to the server

```
} else if (text.startsWith("/name")) { //rra23 10-17-24
    myData.setClientName(text.replace("/name", "").trim());
    System.out.println(TextFX.colorize("Set client name to " + myData.getClientName(), Color.CYAN));
    return true;
```

code that processes command which sets the clients name

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:

The two needed commands are /name and /connect localhost:3000.The code that connects the client usees the address (localhost) and the port (3000) to create an Input and Output Stream that is able to communicate with the server. And the /name command sets the clients name and prints it to the screen in green when they join a lobby

End of Task 2

End of Task 2

End of Group: Start Up
Task Status: 2/2

Group

100%

Group: Communication
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Communication
Task #1: Communication
Weight: ~50%
Points: ~1.50

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

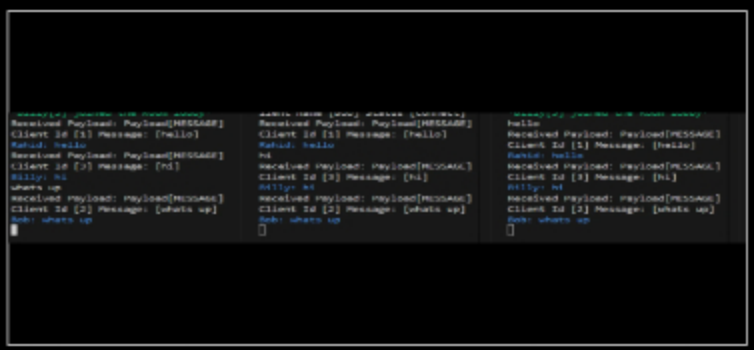
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



3 clients sending and receiving messages from each other

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

```java
//rra23 10-17-24
private void sendMessage(String message) {
    Payload p = new Payload();
    p.setPayloadType(PayloadType.MESSAGE);
    p.setMessage(message);
    send(p);
}
```

client code for sending message over socket

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 code is creating a payload that is of the type MESSAGE. It is then setting the message to whatever it is the user typed. Then it is sending the payload over the socket.

Sub-Task

100%

Group: Communication

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

```java
public boolean sendMessage(long senderId, String message) { //rra23 10-17-24
    Payload p = new Payload();
    p.setClientId(senderId);
    p.setMessage(message);
    p.setPayloadType(PayloadType.MESSAGE);
    return send(p);
}
```

Server relaying messages to each connected client

```java
protected void processPayload(Payload payload) { //rra23 10-17-24
    try {
        switch (payload.getPayloadType()) {
            case CLIENT_CONNECT:
                ConnectionPayload cp = (ConnectionPayload) payload;
                setClientName(cp.getClientName());
                break;
            case MESSAGE:
                currentRoom.sendMessage(this, payload.getMessage());
                break;
```

Server processing message relays to client

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 server is receiving the payload from the client and getting the information from the payload, then it is sending it back to all of the clients that are connected.

Sub-Task

100%

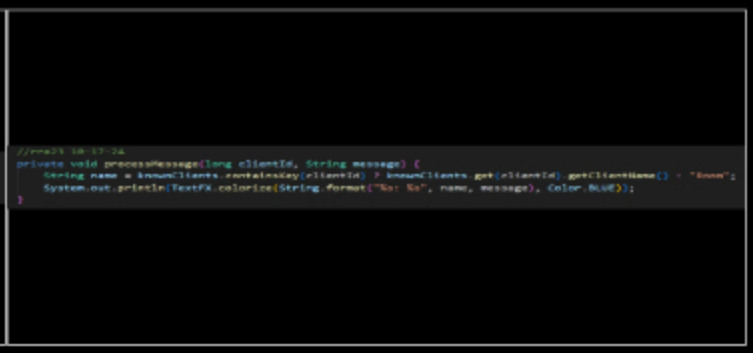
Group: Communication
Task #1: Communication
Sub Task #4: Show the code related to the Client receiving messages from the Server-side and presenting them

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client receiving messages and presenting them



processMessage method that deals with presenting the sender and message

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:

If the client receives a payload that is of type MESSAGE it it will process that message using the processMessage method with the payload sender clientID and payload message as parameters

End of Task 1

Task

100%

Group: Communication
Task #2: Rooms
Weight: ~50%
Points: ~1.50

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 100%

Group: Communication
Task #2: Rooms
Sub Task #1: Show Clients can Create Rooms

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
/createroom TESTING
Received Payload: Payload[ROOM_JOIN
] Client Id [2] Message: [lobby] Cl
ient Name [Bob] Status [disconnect]
*Bob[2] left the Room lobby*
Received Payload: Payload[ROOM_JOIN
] Client Id [2] Message: [TESTING]
Client Name [Bob] Status [connect]
*Bob[2] joined the Room TESTING*
```

client creating a room

Caption(s) (required) ✓

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

Sub-Task 100%

Group: Communication
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

```
/joinroom TESTING
Received Payload: Payload[ROOM_JOIN] Clien
t Id [3] Message: [lobby] Client Name [Bil
ly] Status [disconnect]
*Billy[3] left the Room lobby*
Received Payload: Payload[ROOM_JOIN] Clien
t Id [3] Message: [TESTING] Client Name [B
illy] Status [connect]
*Billy[3] joined the Room TESTING*
Received Payload: Payload[SYNC_CLIENT] Cli
ent Id [2] Message: [null] Client Name [Bo
b] Status [connect]
```

client joining an existing room

Caption(s) (required) ✓

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

Sub-Task 100%

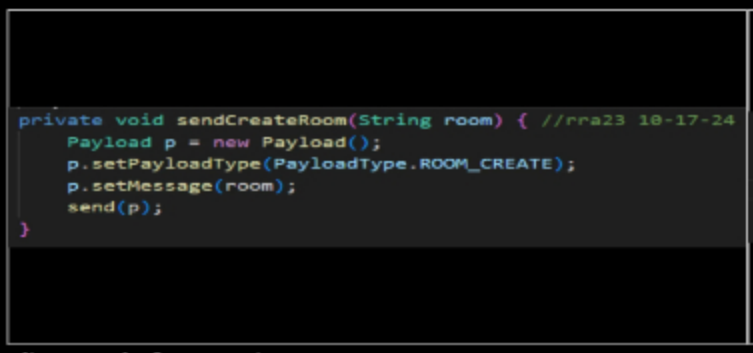
Group: Communication
Task #2: Rooms

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

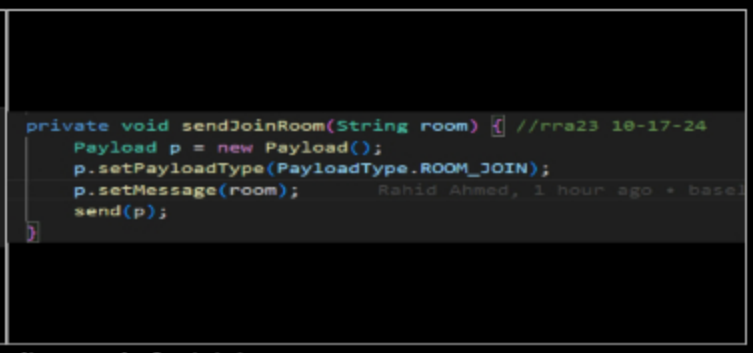
Task Screenshots

Gallery Style: 2 Columns

4 2 1



client code for creating room



client code for joining a room

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 sendCreateRoom method sends a payload with the type ROOM_CREATE, which lets the server know that the client wants to create a new room and be placed in it.

The sendJoinRoom method sends a payload with the type ROOM_JOIN which lets the server know that the client wants to join a room that already exists.

Sub-Task

100%

Group: Communication

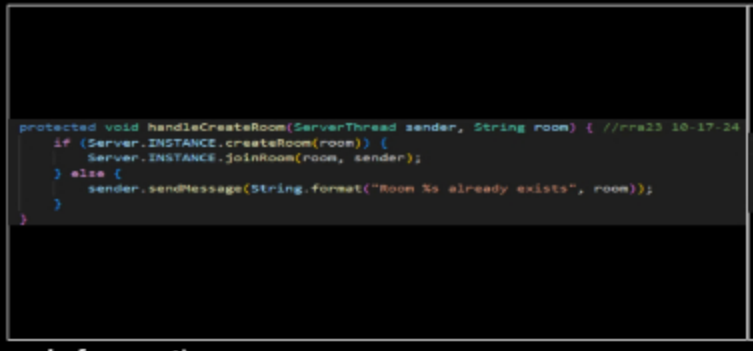
Task #2: Rooms

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



code for creating a new room

code for joining a room

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 handleCreateRoom method creates a room and uses places the client in the room that they created, if the room that they are trying to create already exists than the client will get a message saying that the room already exists.

The handleJoinRoom method puts the client in the room that they want to join and if the room that they typed does not exist it sends a messages saying that it does not exist.

Sub-Task

100%

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 createRoom(String name) { //rra23 10-17-24
    final String nameCheck = name.toLowerCase();
    if (rooms.containsKey(nameCheck)) {
        return false;
    }
    Room room = new Room(name);
    rooms.put(nameCheck, room);
    System.out.println(String.format("Created new Room %s", name));
    return true;
}
```

Server code for creating rooms

```
protected boolean joinRoom(String name, ServerThread client) { //rra23 10-17-24
    final String nameCheck = name.toLowerCase();
    if (!rooms.containsKey(nameCheck)) {
        return false;
    }
    Room current = client.getCurrentRoom();
    if (current != null) {
        current.removedClient(client);
    }
    Room next = rooms.get(nameCheck);
    next.addClient(client);
    return true;
}
```

server code for joining a room

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 createRoom method takes in a room name, changes it to lowercase and places it in the variable called nameCheck. If there is already a room with the same name, the method will return false indicating that the room was not created. If the room name is unique, it will create a new room and add it to the tracked rooms collection and return true.

The joinRoom method checks to see if the room is in the rooms collection, if it is not, it will return false indicating that the room join failed. If the client is currently in a room, it will remove it from that one and add it to the one that it wants to join.

Sub-Task

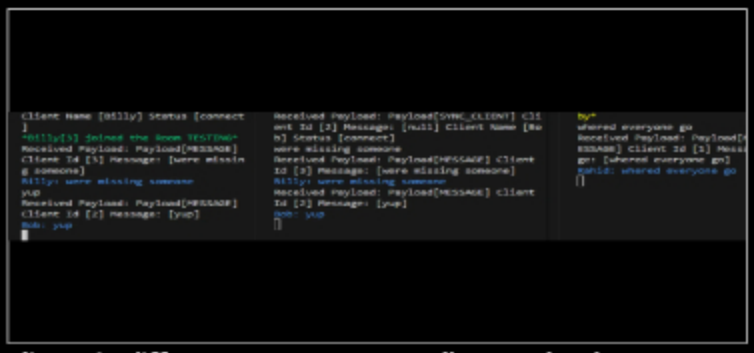
100%

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

Gallery Style: 2 Columns

4 2 1



clients in different rooms cannot talk to each other

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain why/how it works this way

Response:

Since the clients are in different rooms, the payloads that are sent from a person in a room are only sent to people that are in the same room. Also the payloads sent from anywhere outside of the room are not sent to the people in the room.

End of Task 2

End of Group: Communication
Task Status: 2/2

Group

100%

Group: Disconnecting/Termination
Tasks: 1
Points: 3

COLLAPSE

Task

100%

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)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 1

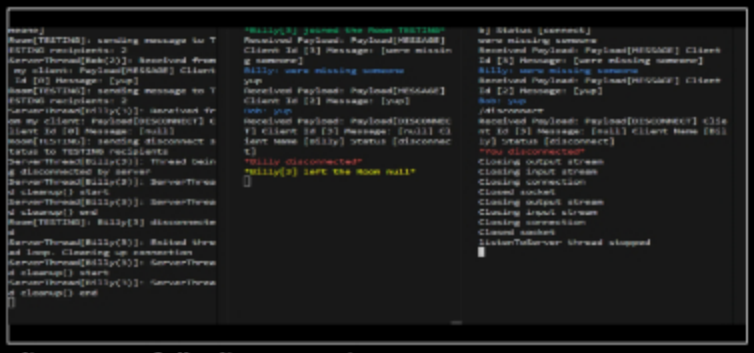
Sub-Task 100%

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



client gracefully disconnecting

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

```
private void sendDisconnect() { //rra23 10-17-24
    Payload p = new Payload();
    p.setPayloadType(PayloadType.DISCONNECT);
    send(p);
}
```

```
private void processDisconnect(long clientId, String clientName) {  //rra23 10-17-24
    System.out.println(
            Textfx.colorize(String.format("%s disconnected*",
                    clientId == myData.getClientId() ? "You" : clientName),
                    Color.RED));
    if (clientId == myData.getClientId()) {
        closeServerConnection();
    }
}
```

code that tells server that a client wants to disconnect

code that deals with other clients disconnecting

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 sendDisconnect method sends a paylaod that tells the server that a client wants to disconnect from it.

The processDisconnect is used to show other clients when a client disconnects. If the client that disconnected runs this method it will also close their server connection.

Sub-Task

100%

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

server disconnecting

Caption(s) (required) ✓

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

Sub-Task

100%

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

```
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();
    }
}
```

Server termination

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 shutdown method removes all of the rooms from the rooms collection and disconnects all of the clients that are in each room.

End of Task 1

End of Group: Disconnecting/Termination
Task Status: 1/1

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Misc
Task #1: Add the pull request link for this branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Task URLs

URL #1
https://github.com/Rahid-Ahmed/rra23-IT114-003/pull/11

URL
https://github.com/Rahid-Ahmed/rra23-IT114-00

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:

Few related sentences about the Project/sockets topics

Task Response Prompt

Response:

I did not have any issues while completing the assignment.

I learned the basic functions of the client and server interaction. I also learned about which code does what. I also learned how to make sure that all of the files are working.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

COLLAPSE

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

overall

detailed view

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment