Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-module-5-project-milestone-1/grade/bs679

Course: IT114-005-F2024
Assigment: [IT114] Module 5 Project Milestone 1
Student: Brandon S. (bs679)

Submissions:

Submission Selection

1 Submission [submitted] 10/29/2024 2:14:29 AM

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

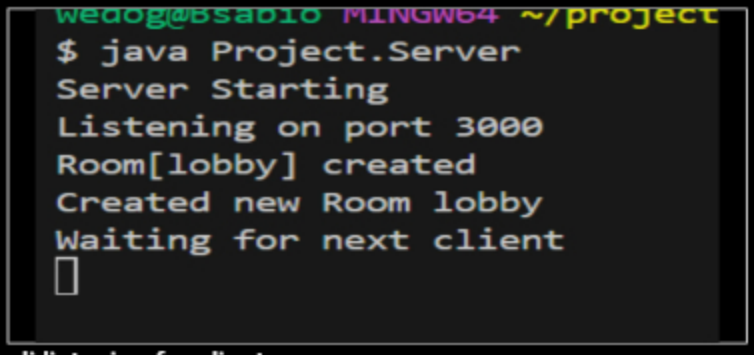
100%

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



cli listening for clients

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

the code that listens for the 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 code first gets the port from the parameter and brings it into the code using this.port = port. A room is created and a while loop is used to wait for the first connection of the client. A blocking action is used to keep the thread from running until a client connects to the room

Sub-Task

100%

Group: Start Up

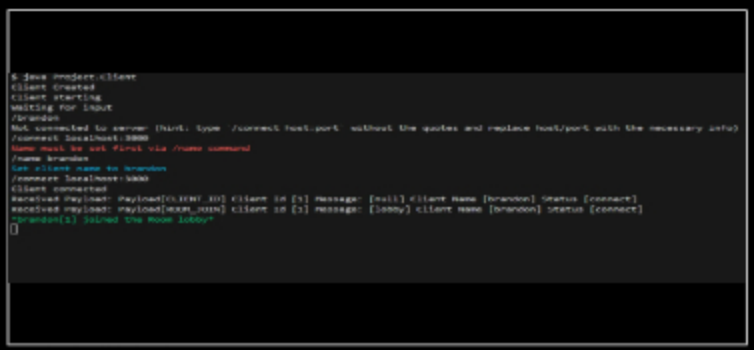
Task #1: Start Up

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



client connecting

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Start Up

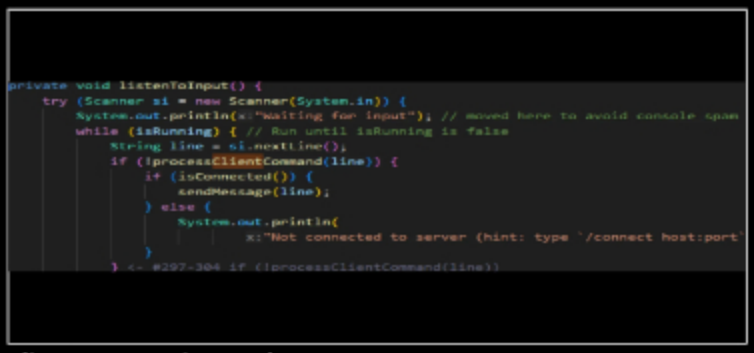
Task #1: Start Up

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



```
private void listenToInput() {
    try (Scanner si = new Scanner(System.in)) {
        System.out.println(x:"Waiting for input"); // moved here to avoid console span
        while (isRunning) { // Run until isRunning is false
            String line = si.nextLine();
            if (!processClientCommand(line)) {
                if (isConnected()) {
                    sendMessage(line);
                } else {
                    System.out.println(
                            x:"Not connected to server (hint: type `/connect host:port`
                    )
            } <- #297-304 if (!processClientCommand(line))
```

client connecting code

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:

in the code, it is under the process client command function. The code checks if the user had put in the proper data. If they haven't, it will direct them to do so, getting them ready to connect to the server.

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

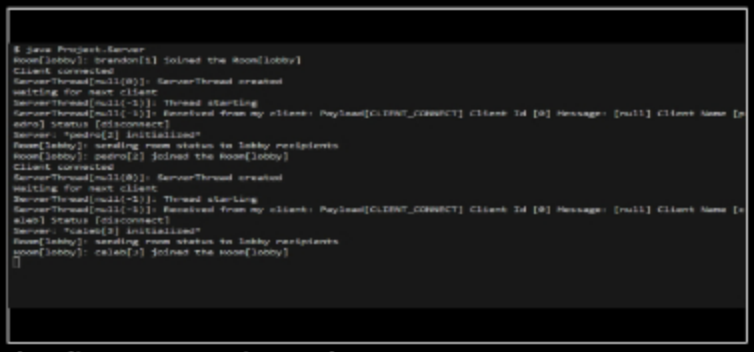
100%

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



the clients connecting to the server

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Start Up

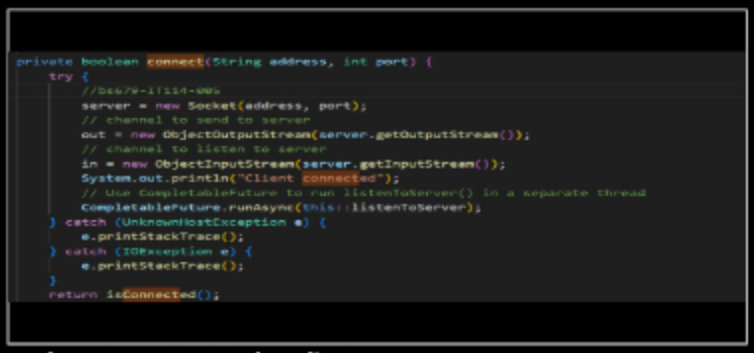
Task #2: Connecting

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



```
private boolean connect(String address, int port) {
    try {
        //Dss7W-17114-WWs
        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();
```

code to connect to the 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

Response:

In the function connect, it takes two parameters, address and port. the server is assigned a port and when the client is detected, the print statement comes out. completeable future is used so that the server can listen for a user while waiting for someone to connect. When the user is detected, it will return disconnected().

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

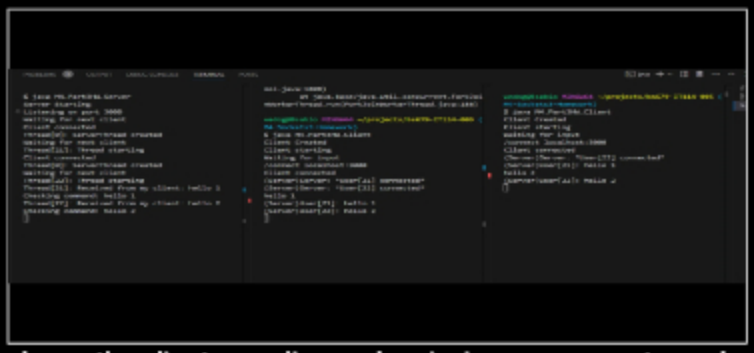
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows the clients sending and recieving messages to each other

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Communication
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

client side sending and receiving code

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:

in the code shown above, the processClientData method checks if the data matches the client data, and if it does ir will add it to the client id and name. In the processMessage method checks if the correct data is there so that the data can be sent out. Once the data is gathered, it is colored blue and formatted.

Sub-Task

100%

Group: Communication

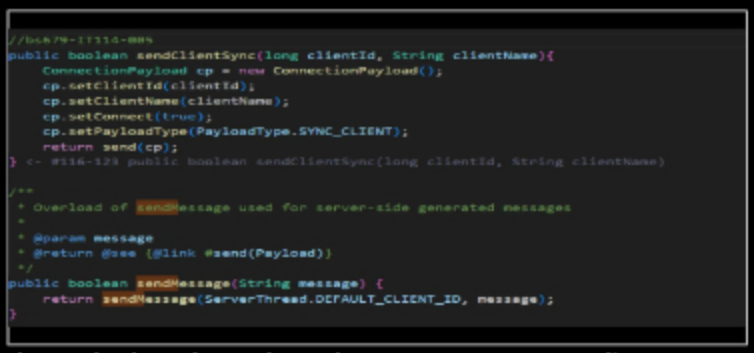
Task #1: Communication

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



the code that shows how the server processes client messages

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:

in the set client sync method, it sets the data so that all the clients will receive the data. In the sendMessage method, it returns the message so that it will be able to sned the message out on the thread.

Sub-Task

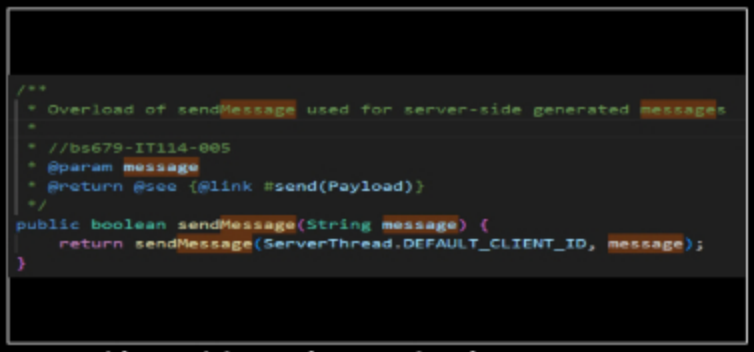
100%

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



```
/**
 * Overload of sendMessage used for server-side generated messages
 *
 * //bs679-IT114-005
 * @param message
 * @return @see {@link #send(Payload)}
 */
public boolean sendMessage(String message) {
    return sendMessage(ServerThread.DEFAULT_CLIENT_ID, message);
}
```

server side receiving and presenting the messages

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:

this blcok of code reseives the message and returns the message using the sendMessage method

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

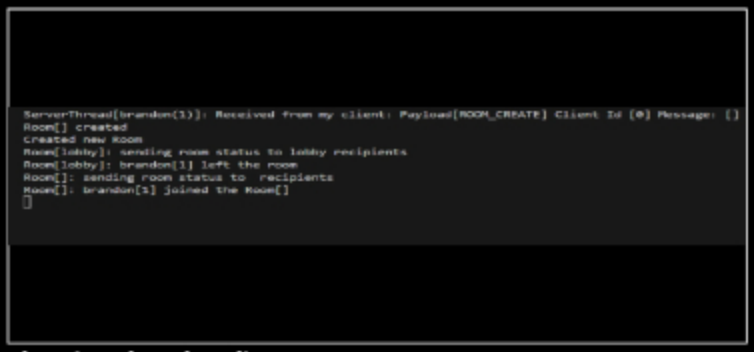
Sub-Task 100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



showing that the client can create a room

Caption(s) (required) ✓

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

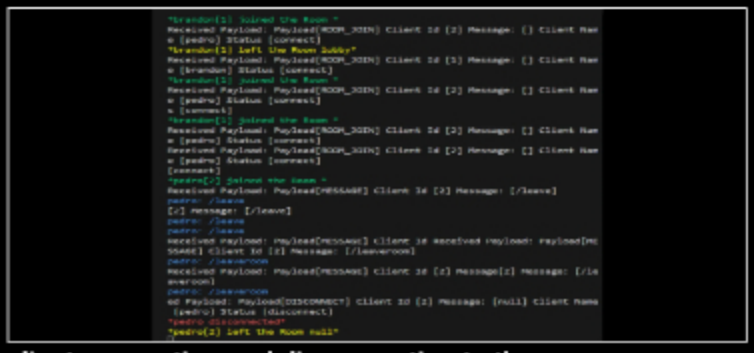
Sub-Task 100%

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



client connecting and disconnecting to the room

Caption(s) (required) ✓

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

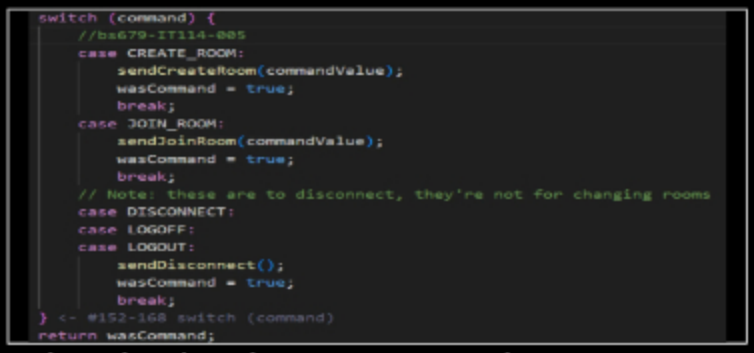
Sub-Task 100%

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



```
switch (command) {
    //bs679-IT114-005
    case CREATE_ROOM:
        sendCreateRoom(commandValue);
        wasCommand = true;
        break;
    case JOIN_ROOM:
        sendJoinRoom(commandValue);
        wasCommand = true;
        break;
    // Note: these are to disconnect, they're not for changing rooms
    case DISCONNECT:
    case LOGOFF:
    case LOGOUT:
        sendDisconnect();
        wasCommand = true;
        break;
} <- #152-168 switch (command)
return wasCommand;
```

code to show how the room was created

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:

this program uses a switch statement so that the program can run down the cases for each command. once the condition is met, it is set to true and the command ends.

Sub-Task

100%

Group: Communication

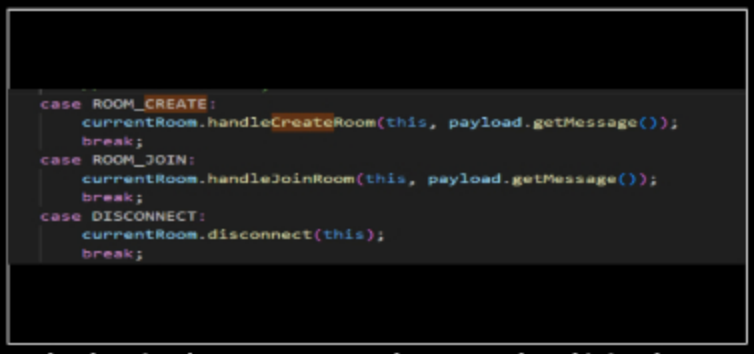
Task #2: Rooms

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
case ROOM_CREATE:
    currentRoom.handleCreateRoom(this, payload.getMessage());
    break;
case ROOM_JOIN:
    currentRoom.handleJoinRoom(this, payload.getMessage());
    break;
case DISCONNECT:
    currentRoom.disconnect(this);
    break;
```

code showing how a room can be created and joined

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:

In this block of code, case statements are used again. When the case is met, it handles the data and then breaks so that the program no longer runs and goes onto the next case.

that the program no longer runs and goes onto the next case.

Sub-Task

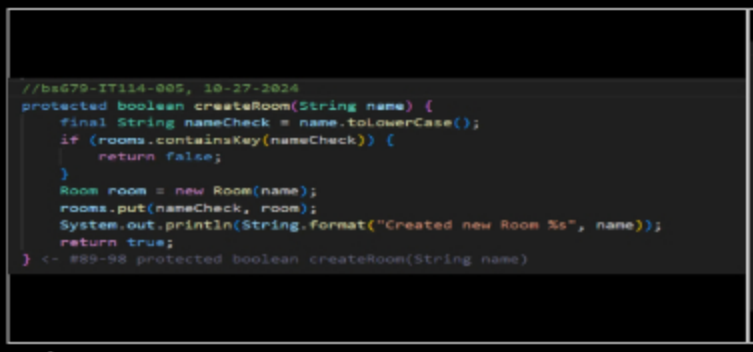
100%

Group: Communication
Task #2: Rooms
Sub Task #5: Show the Server code for handling the create/join process

Task Screenshots

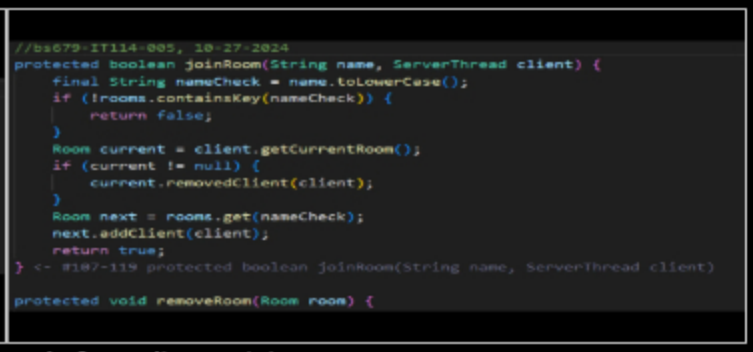
Gallery Style: 2 Columns

4 2 1



```
//bs679-IT114-005, 10-27-2024
protected boolean createRoom(String name) {
    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;
} <- #89-98 protected boolean createRoom(String name)
```

code to create a room



```
//bs679-IT114-005, 10-27-2024
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.removedClient(client);
    }
    Room next = rooms.get(nameCheck);
    next.addClient(client);
    return true;
} <- #107-119 protected boolean joinRoom(String name, ServerThread client)

protected void removeRoom(Room room) {
```

code for a client to join 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:

in the fuction createRoom, the final string changese the name to all lowercase and chcecks if the name exists, and if it doesn't it returns false. Then the code belows creates the room and returns true. In the method joinRoom, it checks if the client already has the id, and it they don't it will return false. Once it returns false, it will change the current room and remove client from the current room. Then the client will be added to the requested room

Sub-Task

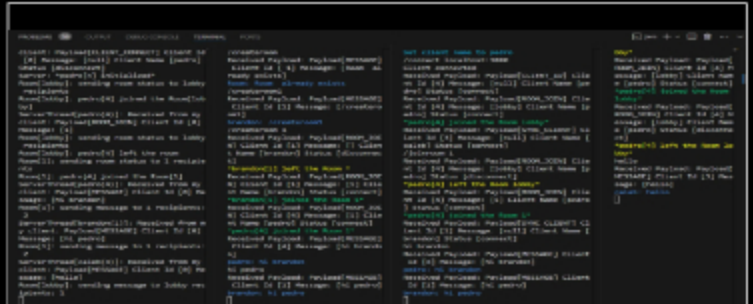
100%

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 talking in seperate rooms

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain why/how it works this way

Response:

this works because each room has their own id, and when the cleint puts in the name of the id, it joins 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

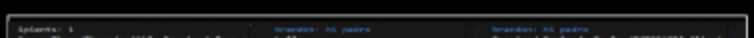
100%

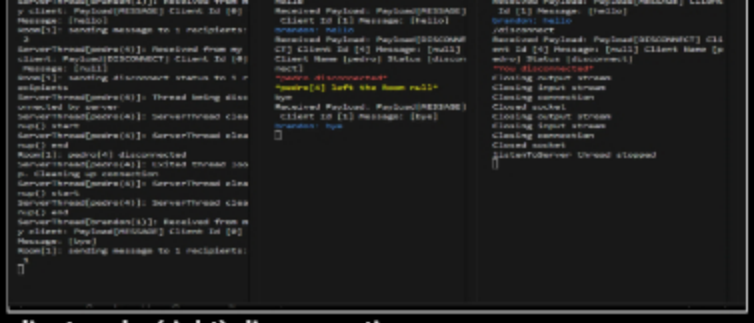
Group: Disconnecting/Termination
Task #1: Disconnecting
Sub Task #1: Show Clients gracefully disconnecting (should not crash Server or other Clients)

Task Screenshots

Gallery Style: 2 Columns

4 2 1





client pedro(right) disconnecting

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Disconnecting/Termination

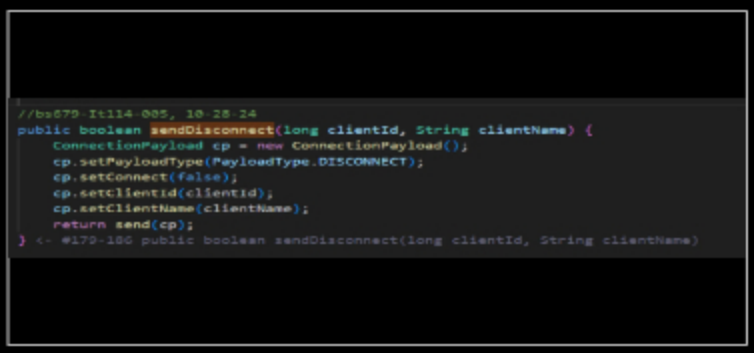
Task #1: Disconnecting

Sub Task #2: Show the code related to Clients disconnecting

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//bs679 It114 005, 10-28-24
public boolean sendDisconnect(long clientId, String clientName) {
    ConnectionPayload cp = new ConnectionPayload();
    cp.setPayloadType(PayloadType.DISCONNECT);
    cp.setConnect(false);
    cp.setClientId(clientId);
    cp.setClientName(clientName);
    return send(cp);
} <- #179-186 public boolean sendDisconnect(long clientId, String clientName)
```

code disconnecting the 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:

In the method

Sub-Task

100%

Group: Disconnecting/Termination

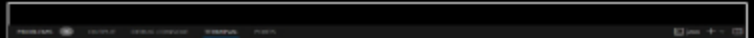
Task #1: Disconnecting

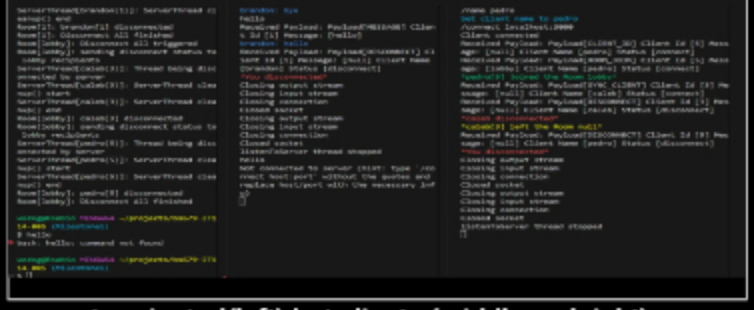
Sub Task #3: Show the Server terminating (Clients should be disconnected but still running)

Task Screenshots

Gallery Style: 2 Columns

4 2 1





server terminated(left) but clients (middle and right) are stull up

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

code of server handling 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:

In the private class Server, it adds something to shut down the thread of the server. Then it will throw out the print statement and use the cleanup meathod.

End of Task 1

End of Group: Disconnecting/Termination

Task Status: 1/1

Group

Group: Misc

Tasks: 3

100%

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/bsabio/bs679-IT114-005/pull/9

URL
https://github.com/bsabio/bs679-IT114-005/pull/

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:

the only problem that I had with this assignment is that it came up during midterms, and it make it hard for me to get done, hence why I am turning this in a week late

End of Task 2

Task

100%

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

COLLAPSE

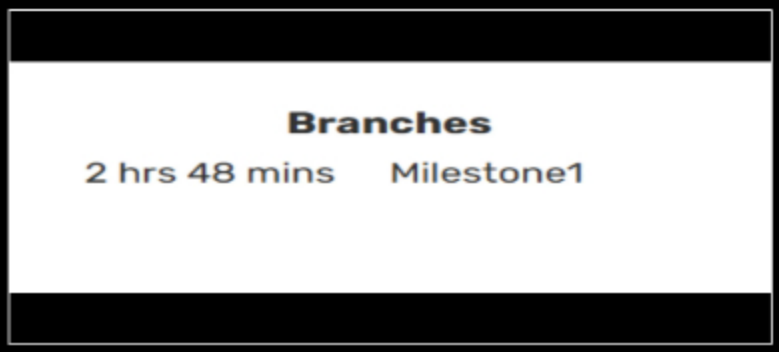
Details:
Grab a snippet showing the approximate time involved that clearly shows your repository.

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



wakatime

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment