

Submission Worksheet

Submission Data

Course: IT114-450-M2025

Assignment: IT114 Milestone 1

Student: Daniel C. (dvc2)

Status: Submitted | **Worksheet Progress:** 100%

Potential Grade: 10.00/10.00 (100.00%)

Received Grade: 0.00/10.00 (0.00%)

Started: 7/14/2025 9:25:11 PM

Updated: 7/14/2025 11:12:26 PM

Grading Link: <https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/grading/dvc2>

View Link: <https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-milestone-1/view/dvc2>

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

```
@dcarch2 → /workspaces/dvc2-it114-450 (Milestone1) $ java M5.Part5.Server
Server Starting
Server: Listening on port 3000
Room[lobby]: Created
Server: Created new Room lobby
Server: Waiting for next client
```

terminal

```
public static void main(String[] args) {
    System.out.println("Server Starting");
    Server server = Server.INSTANCE;
    int port = 3000;
    try {
        port = Integer.parseInt(args[0]);
    } catch (Exception e) {
        // can ignore, will either be index out of bounds or type mismatch
        // will default to the defined value prior to the try/catch
    }
    server.start(port);
    System.out.println("Server Stopped");
```

code



Saved: 7/14/2025 9:29:12 PM

☒ Part 2:

Progress: 100%

Details:

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

Your Response:

The server creates a serversocket on port 3000 and enters a loop until a client accepts. The connections are handled separately on ServerThread to manage the communication independently.



Saved: 7/14/2025 9:29:12 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

main terminal

client1

```
@ Edcanchz: ~ / workspaces / dvc2 - 11114 - 498 ( milestone ) $ java MsClient . Client
Client Created
Client starting
Waiting for input
Name?
Name set to STEVE
CONNECT 127.0.0.1 : 49999
Client connected
Connected
Room [ lobby ] You joined the room
Room [ lobby ] bob#3 joined the room
[]
```

client2

```
@ Edcanchz: ~ / workspaces / dvc2 - 11114 - 498 ( milestone ) $ java MsClient . Client
Client Created
Client starting
Waiting for input
Name?
Name set to bob
CONNECT 127.0.0.1 : 50000
Client connected
Connected
Room [ lobby ] You joined the room
Room [ lobby ] []
```

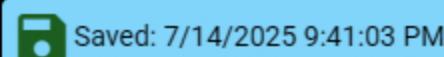
client3

```
private void start ( int port ) {
    this . port = port;
    System . out . println ( "Starting" );
    int i = Server . listening ( port );
    // Create a client connection
    try {
        Client . connect ( port );
        ServerBucket . create ( ServerBucket . class );
        CreateRoom . create ( 10000 );
        // Create the first room ( lobby )
        Room . create ( "lobby" );
        Bucket . incomingClient = serverBucket . accept () ; // blocking action, waits for a client connection
        Client . connected ();
        // Once we receive a connection, pass a callback to notify the server when
        // a player is initialized
        ServerThread . serverThread = new ServerThread ( incomingClient, null );
        serverThread . start (); // This is a threadable utility instance, the lifecycle and we
        // don't have the chance start static
        SERVERPORT . start ();
        // Since we don't yet add the ServerThread reference to our commandClients map
    } catch ( DuplicateBucketException e ) {
        System . out . println ( "Sorry, already exists (this shouldn't happen)" );
        e . printStackTrace ();
    } finally {
        System . out . println ( "Ending server socket" );
    }
}
```

code1

```
3.1 public class ServerThread extends BasicServerThread {
3.2     private Consumer<ServerConnection> onClientIsReady; // callback to inform when this object is ready
3.3
3.4     /**
3.5      * A message method we can short circuit for basic testing with the basic socket source
3.6      * like this:
3.7      * 
3.8      * @param message
3.9      */
3.10     protected void onIncoming ( Message message ) {
3.11         onClientIsReady . accept ( message );
3.12     }
3.13 }
```

code2



Details:

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

Your Response:

The server accepts incoming client connections through a loop. It generates a new ServerThread instance for each connection, which runs concurrently to handle interactions with that particular client.



Saved: 7/14/2025 9:41:03 PM

Section #3: (2 pts.) Feature: Server Will Implement The Concept Of Rooms (With The Default Being "Lobby")

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)

A terminal window displaying the server's log output. The log shows the creation of a 'lobby' room, a client connecting to it, and another client joining the same room. The log also includes messages about room statistics and room removal.

```
Server[lobby] Room created
Client[lobby] Client starting
Waiting for input
/names done
Name set to dan
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] bob2 joined the room
/createroom MyNewRoom
Room[lobby] You left the room
Room[MyNewRoom] You joined the room
```

terminal

A terminal window displaying the client's log output. It shows the client connecting to the server, creating a room named 'MyNewRoom', and joining that room.

```
@ EdgarsPC2 ~>/workspaces/dvc2-11114-450 [Milestone1] $ java -cp .. MS.Parts.Client
Client Created
Client starting
Waiting for input
/names done
Name set to dan
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] bob2 joined the room
/createroom MyNewRoom
Room[lobby] You left the room
Room[MyNewRoom] You joined the room
```

```
Room[MyNewRoom] bob#12 joined the room
/leave room
Room[MyNewRoom] You left the room
Room[lobby] You joined the room
[]
```

client1

```
⑤ @dcerch2 → /workspaces/dvc2-1t14-450 [Milestone1] $ java -cp . HG_Part5.Client
Client Created
Client Starting
Waiting for input
/Name bob
Name set to bob
/Connect localhost:4444
Client connected
User bob
Room[lobby] You joined the room
Room[lobby] daniel left the room
/joinRoom MyHabitRoom
Room[lobby] You left the room
Room[MyHabitRoom] You joined the room
Room[MyHabitRoom] daniel left the room
[]
```

client2

```
200  
201     public Room createRoom(String name) {  
202         Room room = new Room(name);  
203         rooms.put(name, room);  
204         return room;  
205     }  
206  
207  
208     public Room getRoom(String name) {  
209         return rooms.get(name);  
210     }  
211 }
```

server.java

server.java

```
project wide setting, which is the default. This means that the build system will use the same compiler settings as the project's main configuration. If you want to use different compiler settings for a specific module, you can do so by specifying them in the module's configuration file.
```

room.java

```
2.0.0  // attempt to gracefully close and migrate clients
2.0.1  if (clients.size() > 0) {
2.0.2      Room room = lobby.getRoom();
2.0.3      room.migratingClients();
2.0.4      room.broadcast("migrating to lobby");
2.0.5      room.broadcast("clients left room");
2.0.6      room.broadcast("clients joined lobby");
2.0.7  }
2.0.8
2.0.9  Server.INTERFACE.sendMessage(room.Lobby, client);
2.0.10 } catch (RoomNotFoundException e) {
2.0.11     e.printStackTrace();
2.0.12 }
2.0.13 }
```

room.java



Saved: 7/14/2025 10:25:35 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The Server and Room classes are generally used on the server side to manage room creation, joining, leaving, and removal. The Server manages Room instances globally (creation and removal from its map), whilst individual Room objects maintain their client lists, permit client mobility (add/remove), and implement auto-cleanup logic for self-removal when empty.



Saved: 7/14/2025 10:25:35 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

Code Editor



terminal

```
@darch2:~/workspaces/dvc2-1tii4-450 [Milestone1] $ java -cp . MG.Part5.Client
Client created
Client starting
Waiting for input
/name dan
Name set to dan
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] bob#2 joined the room
/leaveRoom MyNewRoom
Room[lobby] You left the room
Room[MyNewRoom] You joined the room
Room[MyNewRoom] bob#2 joined the room
/leaveRoom
Room[MyNewRoom] You left the room
Room[lobby] You joined the room
Room[lobby] steve#1 joined the room
```

client1

```
○ @dmcnch2 ~/workspace/dvc2-16114-450 (M1testone) $ java -cp .:MS-Part5.Client Client created
Client starting
Waiting for input
/names bob
Name set to bob
/connect localhost:3000
Client connected
Connected
Room[lobby] you joined the room
Room[lobby] daniel left the room
/joinroom MyNewRoom
Room[MyNewRoom] you joined the room
Room[MyNewRoom] daniel left the room
Room[MyNewRoom] daniel left the room
```

client2

```
cd /opt/wkspaces/chvct-4.1.14-420 [Milestone1] $ java -cp .:lib/main-client.jar com.main.Client
Creating Client...
Starting listening for input...
Name set to steve
CONNECT localhost:4200
Client connected
Client disconnected
steve@steve:~$
```

client3

```
494  
495     public static void main(String[] args) {  
496         Client client = Client.INSTANCE;  
497         try {  
498             client.start();  
499         } catch (IOException e) {  
500             System.out.println("Exception from main()");  
501             e.printStackTrace();  
502         }  
503     }  
504 }
```

client.java

```
1.1 package com.ether.chatserver;
1.2 import java.io.BufferedReader;
1.3 import java.io.IOException;
1.4 import java.io.InputStreamReader;
1.5 import java.net.Socket;
1.6 import java.util.Scanner;
1.7
1.8 public class ClientThread extends Thread {
1.9     private ConnectionServer server;
2.0     private Socket socket;
2.1     private Scanner scanner;
2.2     private BufferedReader reader;
2.3     private String message;
2.4
2.5     protected void run() {
2.6         try {
2.7             // Set up connection and get reference to server
2.8             server = new ConnectionServer();
2.9             socket = server.createSocket("127.0.0.1", 5000);
3.0             reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
3.1             scanner = new Scanner(System.in);
3.2             message = scanner.nextLine();
3.3             System.out.println("Connected to server");
3.4             send();
3.5         } catch (IOException e) {
3.6             e.printStackTrace();
3.7         }
3.8     }
3.9
3.10     protected void send() {
3.11         try {
3.12             // Send /name command and get response
3.13             writer.write("/name " + message);
3.14             writer.flush();
3.15             String response = reader.readLine();
3.16             System.out.println(response);
3.17             if (response.equals("Name accepted")) {
3.18                 writer.write("/connect " + message);
3.19                 writer.flush();
3.20                 String response2 = reader.readLine();
3.21                 System.out.println(response2);
3.22             }
3.23         } catch (IOException e) {
3.24             e.printStackTrace();
3.25         }
3.26     }
3.27
3.28     protected void disconnect() {
3.29         try {
3.30             writer.write("/quit");
3.31             writer.flush();
3.32             String response = reader.readLine();
3.33             System.out.println(response);
3.34         } catch (IOException e) {
3.35             e.printStackTrace();
3.36         }
3.37     }
3.38 }
3.39 
```

serverthread.java



Saved: 7/14/2025 10:32:12 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:

Before connecting, you run the /name command to configure the client's username locally. The /connect command then sends a connection request to the server, which accepts it and launches a ServerThread to handle the conversation.



Saved: 7/14/2025 10:32:12 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

terminal

```
glenanchizi → /workspaces/dvclz-13314-4549 (Milestone1) $ java -cp .:MsParts.Client  
Client Created  
Client Starting  
Waiting for input  
/name dan  
Name set to dan  
Connection to address: 127.0.0.1  
Client Connected  
Connected  
Room[lobby] You joined the room  
Room[tobin] bob123 joined the room  
/createRoom MyNewRoom  
Room[MyNewRoom] You joined the room  
Room[MyNewRoom] bob123 joined the room  
Room[MyNewRoom] steve1 joined the room  
/leaveRoom  
Room[tobin] You left the room  
Room[tobin] You joined the room  
Room[lobby] steve1 joined the room  
/joinRoom CoolGuyRoom  
Room[coolguyroom] You joined the room  
Room[coolguyroom] bob123 joined the room  
Room[coolguyroom] steve1 joined the room
```

client 1

```
⇒ @dmcanchz → /workspaces/dvcf-11114-4546 (milestonet) $ java -cp .:MyParts client  
Client Created  
Client starting  
Waiting for input  
/name bob  
Name set to bob  
/connect localhost:2000  
Client connected  
Connected  
Room[lobby] You joined the room  
Room[lobby] daniel left the room  
/join room MyNewRoom  
Room[lobby] you left the room  
Room[MyNewRoom] You joined the room  
Room[MyNewRoom] daniel left the room  
/join room CoolGuyRoom  
Room[MyNewRoom] You left the room  
Room[CoolGuyRoom] You joined the room  
Room[CoolGuyRoom] stevel3 joined the room
```

client 2

```
== @dcarchz: ~ /workspaces/dvcz-it114-454R (milestones) $ java -cp .:MsParts.Client Client Created
Client Starting
Waiting for Input
Name: Steve
Name set to steve
Address: localhost: 66666
Client connected
Client ID:
Room[lobby] you joined the room
Room[lobby] Steve joined the room
/Johnson CoolMyRoom
Room[lobby] You left the room
Room[CoolMyRoom] You joined the room
```

client 3

snippet 1

```

protected void OnDownloadCompleted(DownloadEventArgs e)
{
    if (e.Result == DownloadResult.Success)
    {
        string file = Path.Combine("C:\\", e.FileName);
        File.WriteAllBytes(file, e.Data);
        MessageBox.Show("Download completed successfully!");
    }
    else
    {
        MessageBox.Show("Download failed with error code: " + e.Result);
    }
}

```

snippet 2



Saved: 7/14/2025 10:43:14 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:

On the client side, `processClientCommand` parses the `/createroom` and `/joinroom` commands before using `sendRoomAction` to send a Payload to the server containing the room name and the relevant action type (`ROOM_CREATE` or `ROOM_JOIN`). On the server side, the `ServerThread` received this payload and delegated to the Room's `handleCreateRoom` or `handleJoinRoom` methods, which then used `Server.INSTANCE` to manage the room creation or client's room transfer.



Saved: 7/14/2025 10:43:14 PM

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

Progress: 100%

≡ Task #1 (1 pt.) - Evidence

Progress: 100%



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

```
newRoomComponents: sending message to [client] message: [join]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [client: yo]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: yo]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: yo]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Others: op]
Room[coordinateRoom]: sending message to a recipient: [client]: what's up?
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: what's op]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: what's op]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Others: op]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Others: op]
Room[CoordinateRoom]: sending message to [client] message: [nothing much]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: nothing much]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: nothing much]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Others: op]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Others: op]
Server: Removing [client]. Few people here. Let's continue.
[Thread-0]: sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: leave]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Room[CoordinateRoom]: would like to leave the room]
[Thread-0]: Sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Room[CoordinateRoom]: would like to leave the room]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Room[CoordinateRoom]: You left the room]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
Room[Tobby]: sending message to [client] message: you're back to the lobby
[Thread-0]: sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: yo im back in the lobby]
[Thread-0]: Received from my [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: ClientName [Edmund]]
Room[Tobby]: sending message to [client] message: last alone here
[Thread-0]: sending to [client]: payload[MESSAGE] Client id: [0] Message: [Edmund: all alone here]
```

terminal

terminal

```
    /**
     * Sends a message to the server
     */
    @Param message
    @Throws IOException
    private void sendMessage(String message) throws IOException {
        Payload payload = new Payload();
        payload.setFormattedMessage(message);
        payload.setPayloadType(PayloadType.MESSAGE);
        sendToServer(payload);
    }

    /**
     * Sends the client's name to the server (what the user desires to be called)
     */
    @Param name
    @Throws IOException
    private void sendClientName(String name) throws IOException {
        ConnectionPayload payload = new ConnectionPayload();
        payload.setClientName(name);
        payload.setPayloadType(PayloadType.CLIENT_CONNECT);
        sendToServer(payload);
    }
```

client.java

```
2.26 // and handle() methods
2.27 @Override
2.28 protected void processStepLoad(MessageLoad incoming) {
2.29     switch (IncomingStepType.get(incoming)) {
2.30         case COMMENCEMENT:
2.31             handleCommencement((CommencementIncomingLoad) incoming).process();
2.32         case ENDING:
2.33             handleEnding((EndingIncomingLoad) incoming);
2.34         case MESSAGE:
2.35             handleMessage((MessageIncomingLoad) incoming);
2.36         case REVERSE:
2.37             handleReverse((ReverseIncomingLoad) incoming);
2.38         case REVERSATION:
2.39             handleReversation((ReversationIncomingLoad) incoming);
2.40         case REVERSATION_MESSAGE:
2.41             handleReversationMessage((ReversationMessageIncomingLoad) incoming);
2.42         case REVERSATION_END:
2.43             handleReversationEnd((ReversationEndIncomingLoad) incoming);
2.44         case REVERSATION_MESSAGE_END:
2.45             handleReversationMessageEnd((ReversationMessageEndIncomingLoad) incoming);
2.46     }
2.47     log.info("Received payload type: " + incoming.getPayloadType());
2.48 }
```

serverthread.java

room.java

baseserverthread.java



Saved: 7/14/2025 10:49:36 PM

≡, Part 2:

Progress: 100%

Details:

- Briefly explain how the message code flow works

Your Response:

When a client delivers a message, it is contained in a Payload of type MESSAGE and sent to the server. The server's ServerThread gets this, and the currentRoom uses the relay method to send the message to all clients in that room.



Saved: 7/14/2025 10:49:36 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 clients reconnecting when a server is brought back online
 - Examples should include relevant messages of the actions occurring
 - 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

```
sun.misc.Unsafe
UnsafeAccess
Connection_dropped
java.io.EOFException
java.io.ObjectInputStream$BlockDataInputStream$PeakByte@objectInputStream.java:[222]
at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:[373])
at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:[540])
at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:[540])
at java.util.concurrent.ForkJoinPool$Worker.run(ForkJoinPool.java:[640])
at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:[694])
at java.base/java.lang.Thread.run(Thread.java:[648])
Closing output stream
Closing input stream
Closing connection
[uncaught exception] uncaught exception in thread [main] (runnable: [main])
```

client disconnect

```
Raven[uncaughtErrors]: sending message to 3 residents. Raven[uncaughtErrors]: identified disconnected threads:  
Thread[1]: Thread-Reading-Configuration by someone  
Thread[2]: Server-Thread-Administrator() a user  
Thread[3]: Server-Thread-Administrator() a user  
Thread[4]: GetAndReadThread() a user  
Thread[5]: ReadAndUnreadLog() a user  
Thread[6]: UnreadThread() a user  
Thread[7]: ClearThread() a user  
Thread[8]: ClearThread() a user  
Thread[9]: Server-Thread-Administrator() a user
```

server reaction to client disconnect

server close

client reaction to server close

client reaction to server close

server startup after close

client reconnect after close

client reconnect after close

client reconnect after close

client.java

```
34     */
35     private void shutdown() {
36         try {
37             // chose removeIf over forEach to avoid potential
38             // ConcurrentModificationException
39             // since empty rooms tell the server to remove themselves
40             rooms.values().removeIf(room -> {
41                 room.disconnectAll();
42                 return true;
43             });
44         } catch (Exception e) {
45             e.printStackTrace();
46         }
47     }

```

server.java

```
172
173     protected synchronized void disconnectAll() {
174         info("Disconnect All triggered");
175         if (!isRunning) {
176             return;
177         }
178         clientsInRoom.values().removeIf(client -> {
179             disconnect(client);
180             return true;
181         });
182         info("Disconnect All finished");
183     }
184 }
```

room.java

baseserverthread.java



≡, Part 2:

Progress: 100%

Details:

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

Your Response:

The client gracefully manages disconnection by delivering a specified payload to the server before shutting its own I/O streams and sockets, as well as modifying its internal state and user interface. The server, on the other hand, employs a shutdown hook to loop over all active rooms and their clients, disconnecting each client separately by clearing away their server-side resources and alerting other room participants before ultimately terminating the server.



Saved: 7/14/2025 11:00:17 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%

```
-- Project
-- client
  - Client.java
  - TextX.java
  - User.java
-- Common
  - Command.java
  - ConnectionPayload.java
  - Constants.java
  - Payload.java
  - PayloadType.java
  - Room.java
  - RoomAction.java
-> Exceptions
-- Server
  - BasicServerThread.java
  - Server.java
  - ServerThread.java
-- README.md
```

my workplace structure



Saved: 7/14/2025 11:00:45 PM

- ≡ Task #2 (0.25 pts.) - Github Details

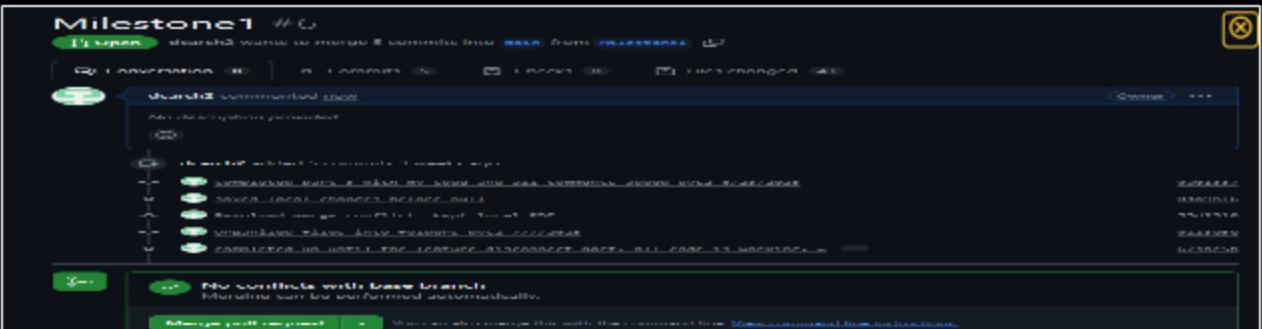
Progress: 100%

❑ Part 1:

Progress: 100%

Details:

From the Commit tab of the Pull Request screenshot the commit history



screenshot



☞ Part 2:

Progress: 100%

Details:

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

URL #1

<https://github.com/dcarch2/dvc2-it114-450>



URL
<https://github.com/dcarch2/dvc2-it114-450>



▣ 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

Projects - dvc2-it114-450

Please enter the last 7 days in `dvc2-it114-450`.



screenshot, but it is still not tracking my proper duration and changes



Saved: 7/14/2025 11:08:34 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:

This assignment taught me the core components and interactions of a client-server chat program. Specifically, I learned how a server maintains multiple client connections utilizing dedicated threads (ServerThread) and orchestrates discussion across separate chat "rooms". The processing of multiple command kinds, such as /name, /connect, /createroom, /joinroom, and /disconnect, became clearer to me, demonstrating how client input is translated into particular server-side operations via Payload objects.



Saved: 7/14/2025 11:10:44 PM

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

Progress: 100%

Details:

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

Your Response:

The easiest part of the assignment, while not necessarily having to do with the actual code, was of course organizing my independent files into the proper subfolders within my main project folder.



Saved: 7/14/2025 11:11:30 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 creating and joining rooms. I was having issues with my terminal displaying the usage of the join and create room commands and my commands were echoing back to me for a while. I was able to fix this but it was definitely the hardest hickup during the assignment.



Saved: 7/14/2025 11:12:26 PM