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 seperately 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

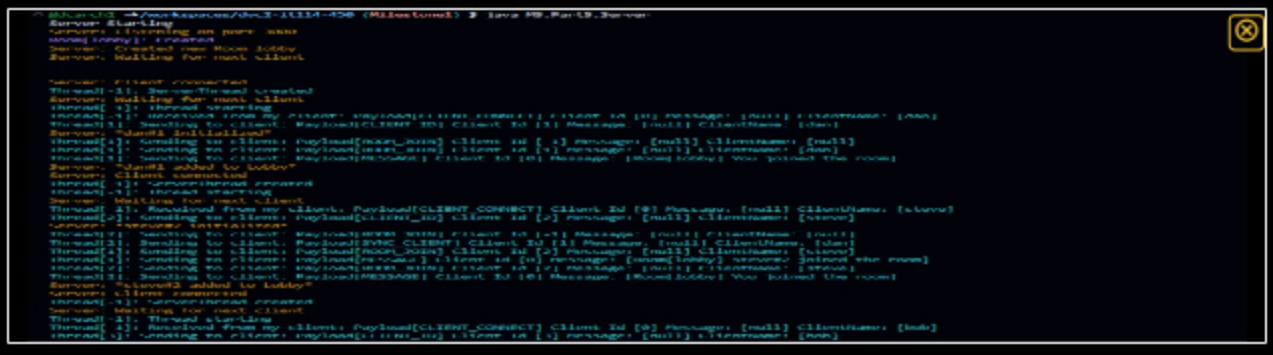
Progress: 100%

Part 1:

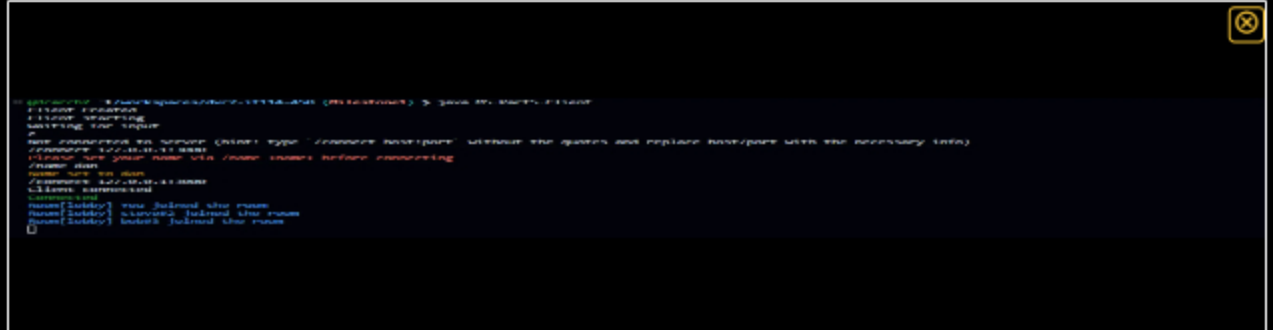
Progress: 100%

Details:

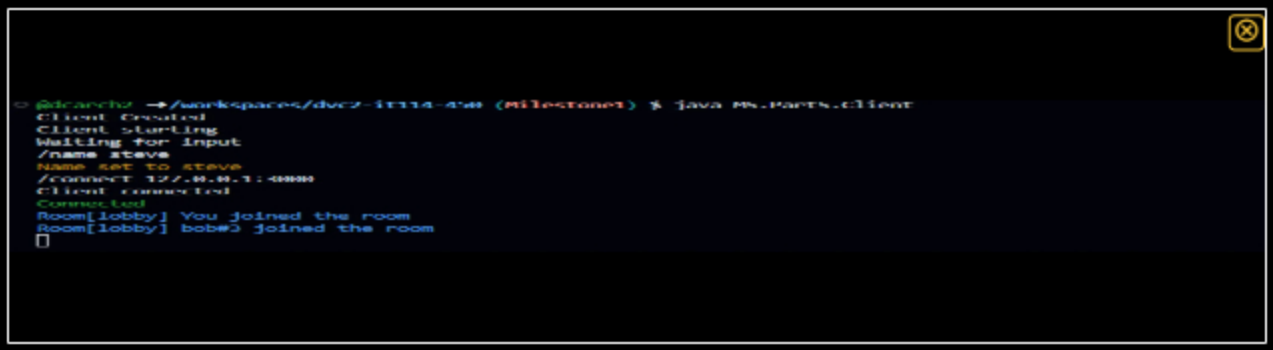
- Show the terminal output of the server receiving multiple connections
- Show at least 3 Clients connected (best to use the split terminal feature)
- Show the relevant snippets of code that handle logic for multiple connections



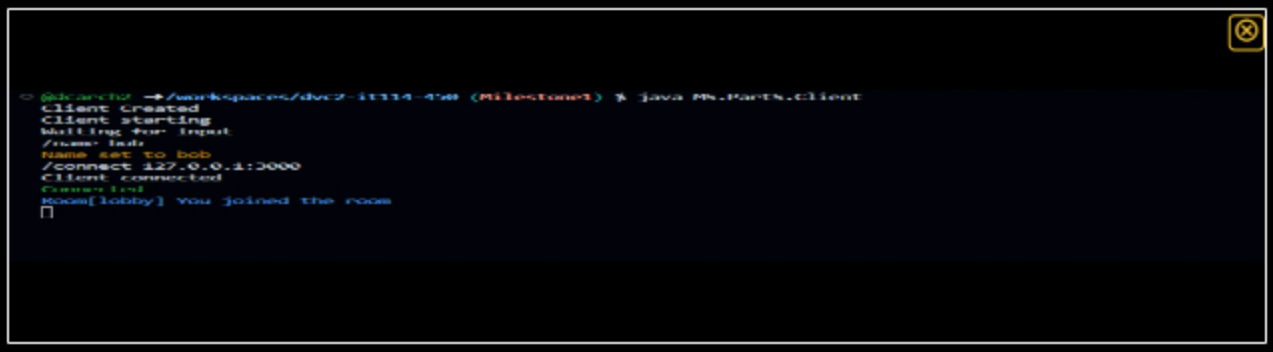
main terminal



client1



client2



client3

code1

code2

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

Part 2:

Progress: 100%

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 "l obby")

Progress: 100%

Task #1 (2 pts.) - Evidence

Progress: 100%

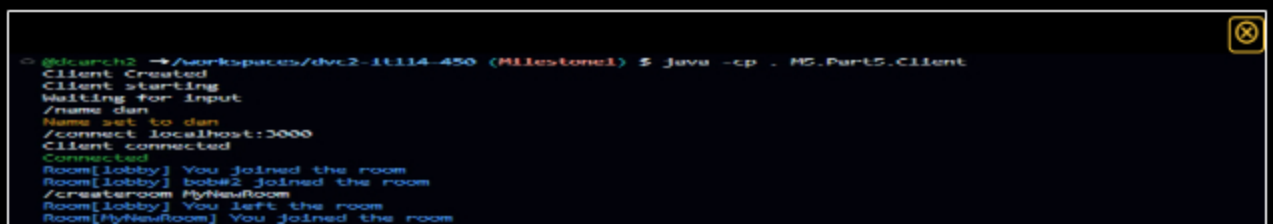
Part 1:

Progress: 100%

Details:

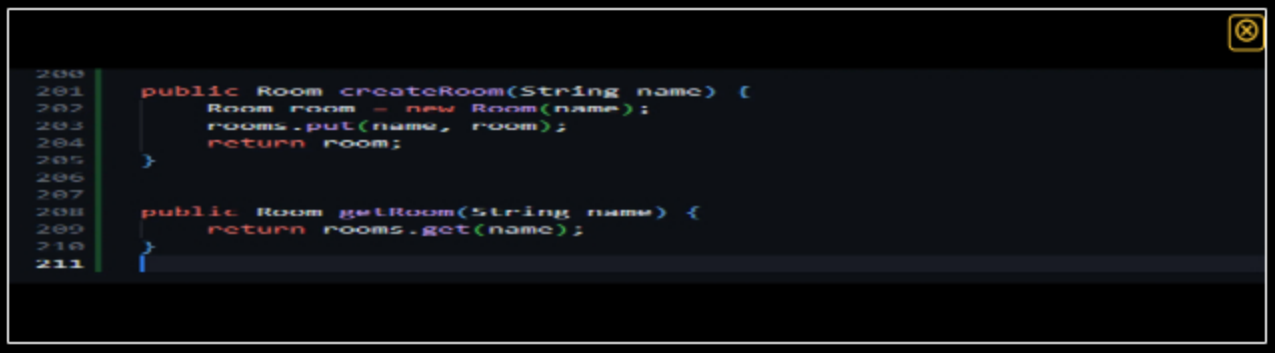
- Show the terminal output of rooms being created, joined, and removed (server-side)
- Show the relevant snippets of code that handle room management (create, join, leave, remove) (server-side)

terminal



client1

client2

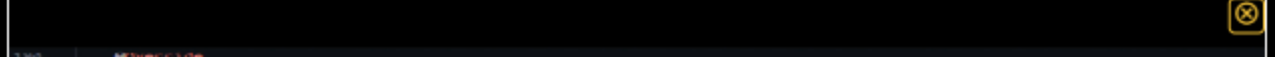


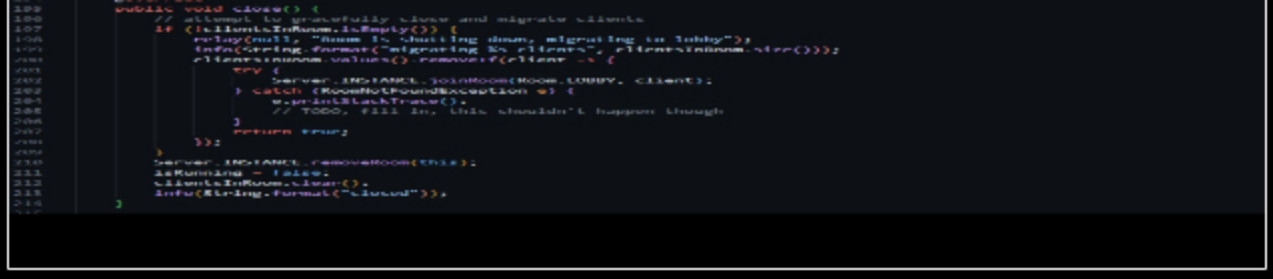
```
200
201     public Room createRoom(String name) {
202         Room room = new Room(name);
203         rooms.put(name, room);
204         return room;
205     }
206
207
208     public Room getRoom(String name) {
209         return rooms.get(name);
210     }
211
```

server.java

server.java

room.java





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

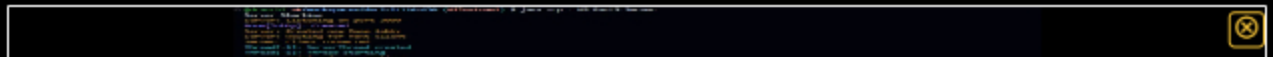
Progress: 100%

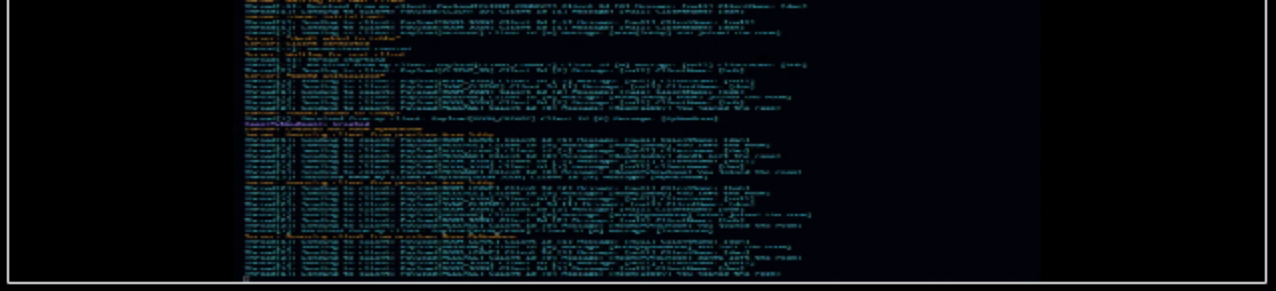
Part 1:

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





terminal

```
@dcarch2 ➜ /workspaces/dvc2-it114-450 (Milestone1) $ java -cp . MS.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
/createroom 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#4 joined the room
```

client1

```
@dcarch2 ➜ /workspaces/dvc2-it114-450 (Milestone1) $ java -cp . MS.Part5.Client
Client Created
Client starting
Waiting for input
/name bob
Name set to bob
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
Room[lobby] dan#1 left the room
/joinroom MyNewRoom
Room[lobby] You left the room
Room[MyNewRoom] You joined the room
Room[MyNewRoom] dan#1 left the room
```

client2

```
@dcarch2 ➜ /workspaces/dvc2-it114-450 (Milestone1) $ java -cp . MS.Part5.Client
Client Created
Client starting
Waiting for input
/name steve
Name set to steve
/connect localhost:3000
Client connected
Connected
Room[lobby] You joined the room
```

client3

```
454
455     public static void main(String[] args) {
456         Client client = Client.INSTANCE;
457         try {
458             client.start();
459         } catch (IOException e) {
460             System.out.println("Exception from main()");
461             e.printStackTrace();
462         }
463     }
464 }
```

client.java

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. Then, 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

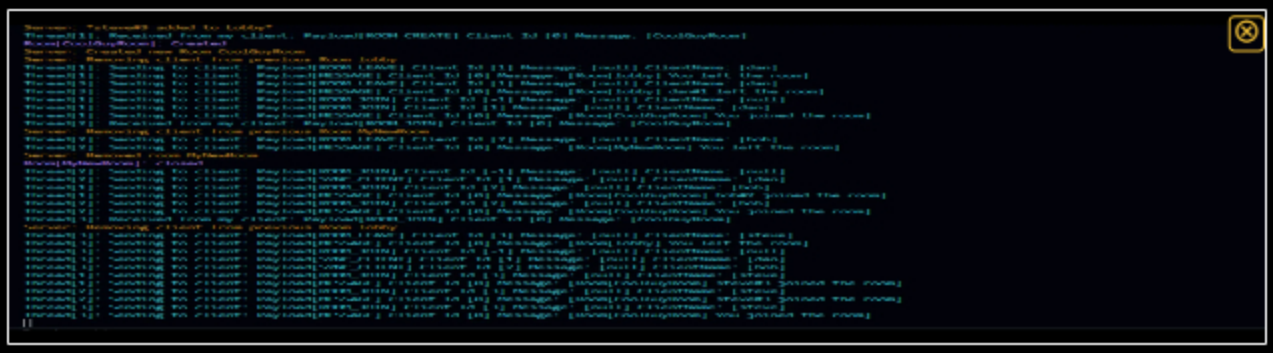
Progress: 100%

Part 1:

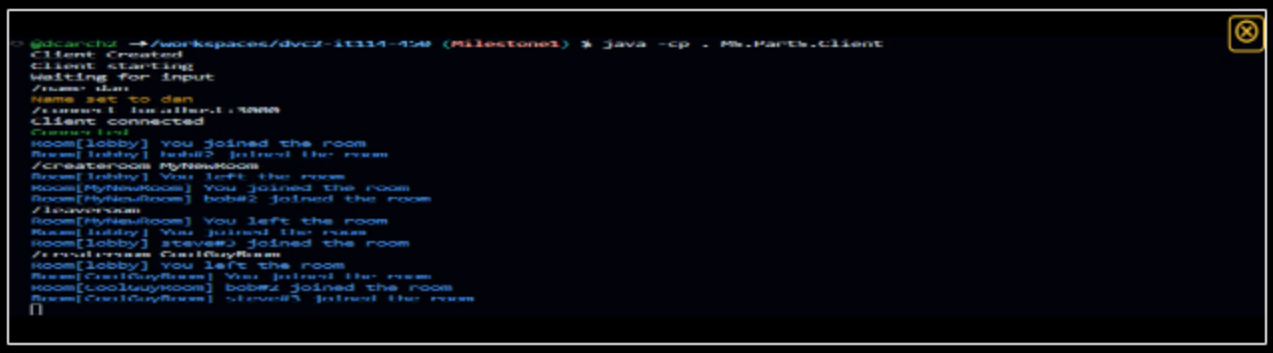
Progress: 100%

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

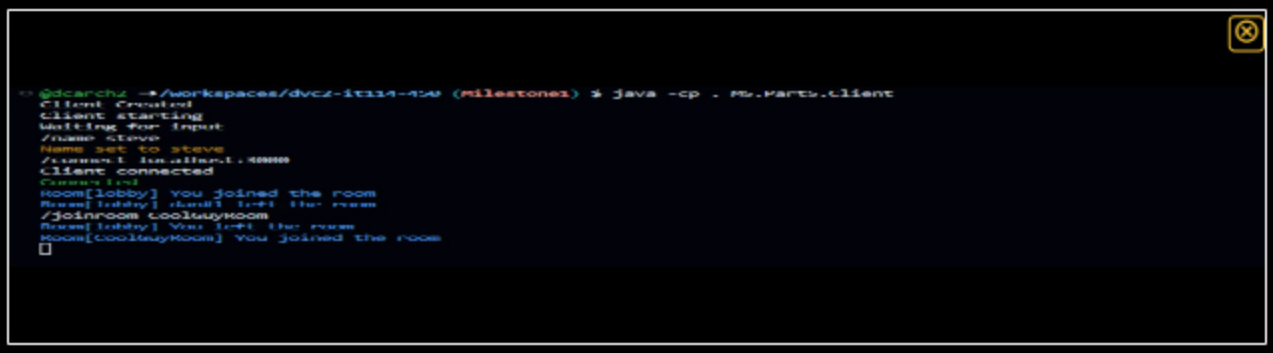


terminal

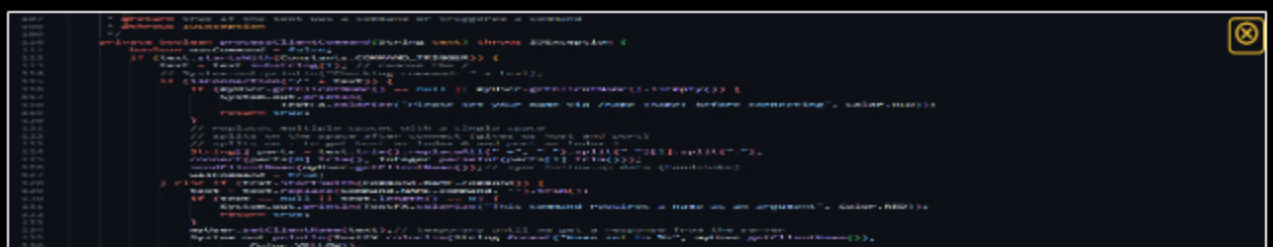


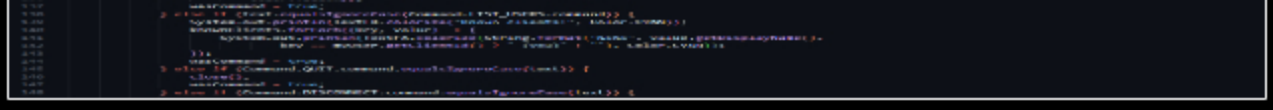
client 1

client 2



client 3





snippet 1

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

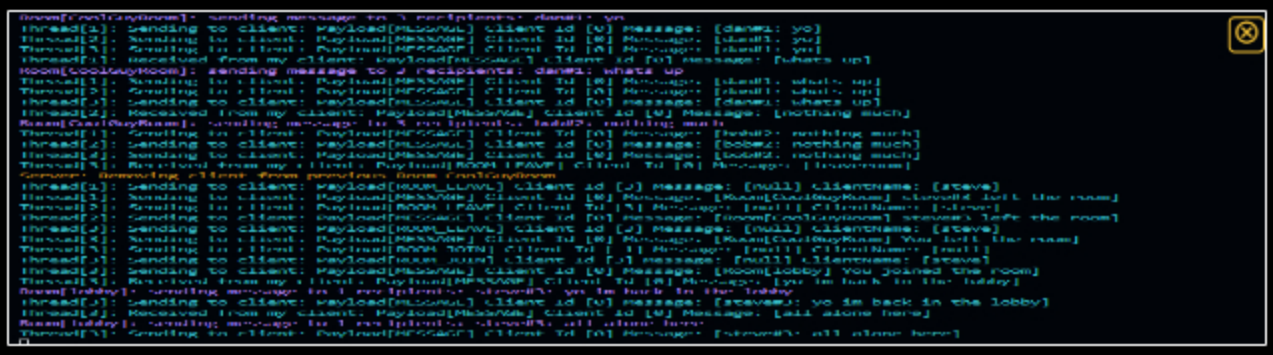
Progress: 100%

Part 1:

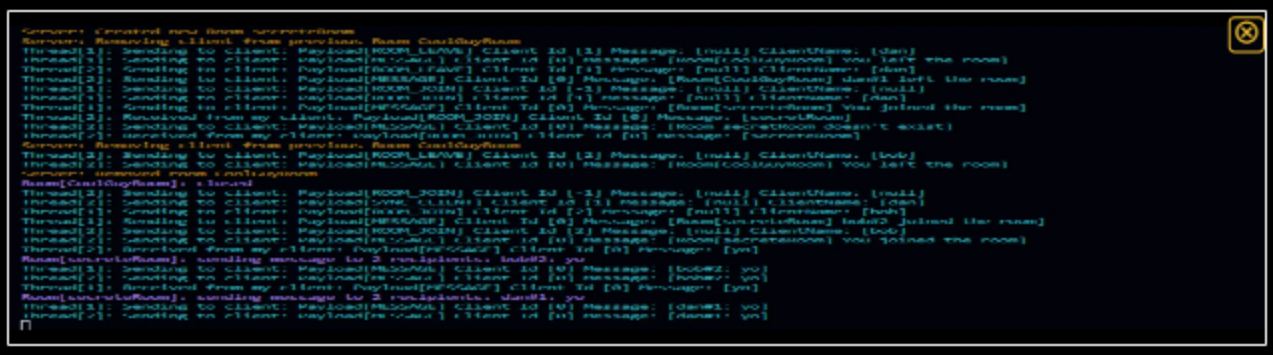
Progress: 100%

Details:

- Show the terminal output of a few messages from each of 3 clients
- Include examples of clients grouped into other rooms
- Show the relevant snippets of code that handle the message process from client to server-side and back



terminal



terminal

client.java

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%

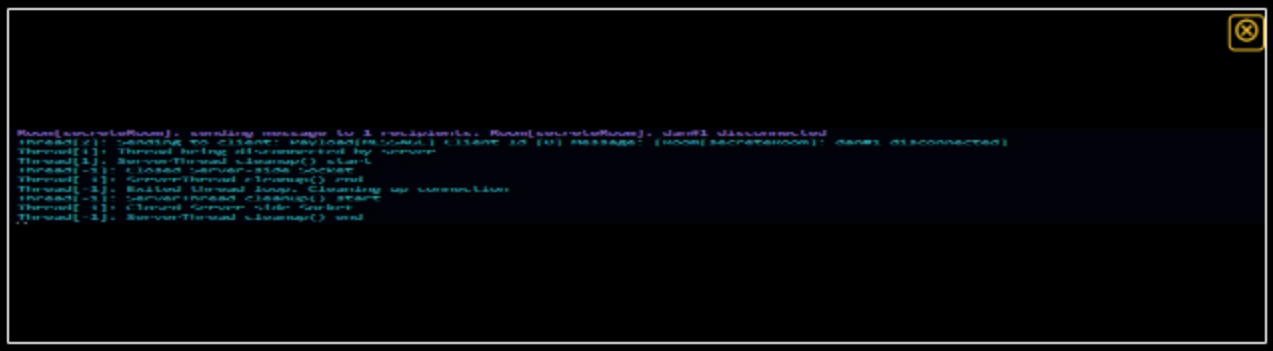
Part 1:

Progress: 100%

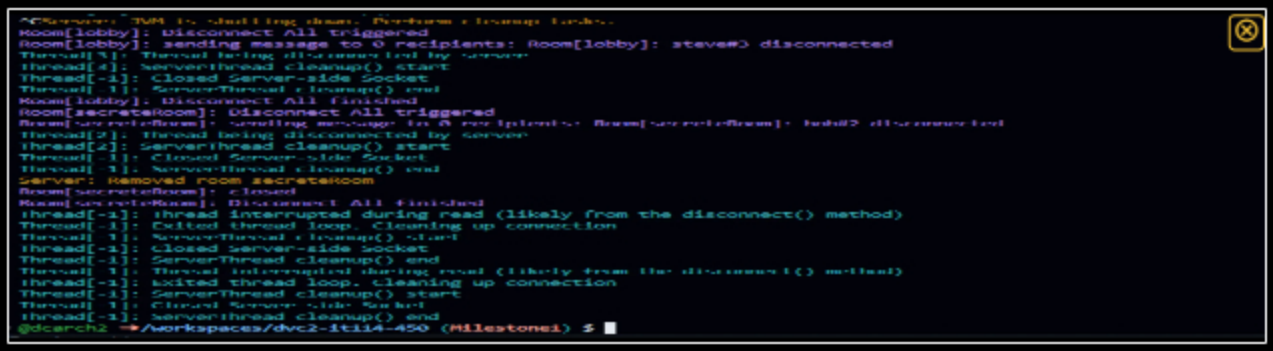
Details:

- 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 occuring
- Show the relevant snippets of code that handle the client-side disconnection process
- Show the relevant snippets of code that handle the server-side termination process

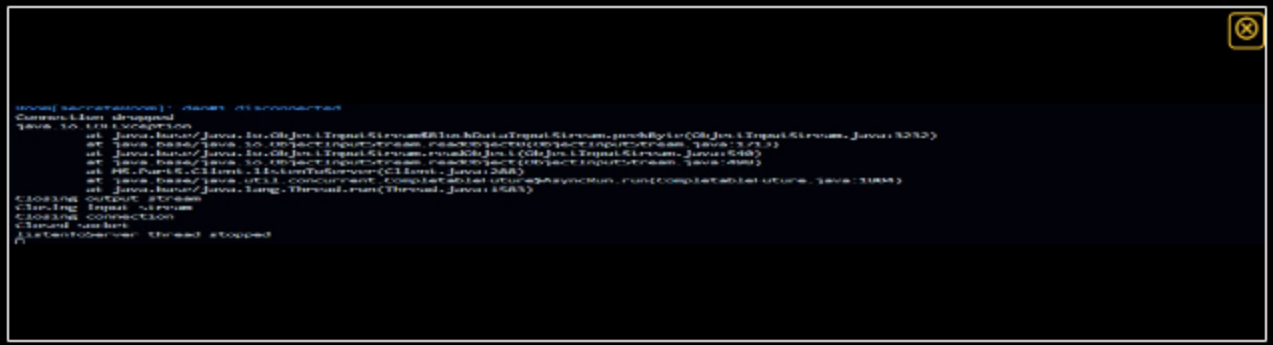
client disconnect



server reaction to client disconnect

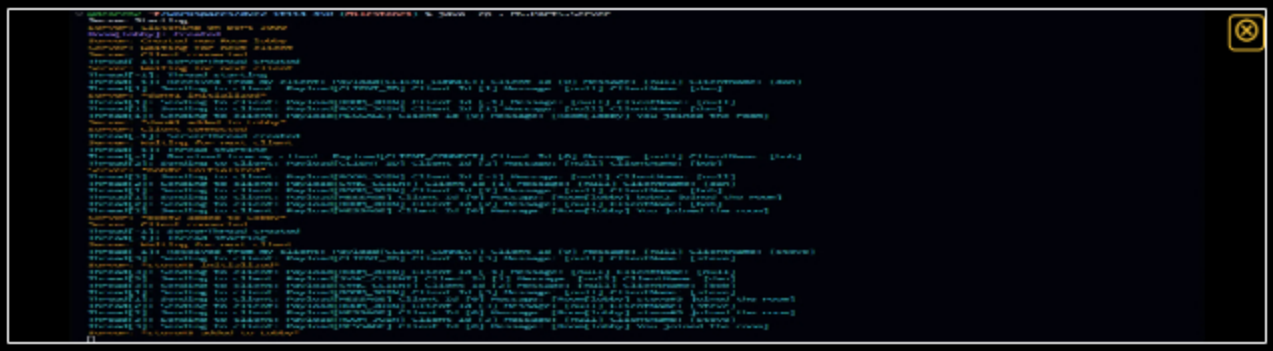


server close



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

```java
     */
    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.java

```java

    protected synchronized void disconnectAll() {
        info("Disconnect All triggered");
        if (!isRunning) {
            return;
        }
        clientsInRoom.values().removeIf(client -> {
            disconnect(client);
            return true;
        });
        info("Disconnect All finished");
    }

```

room.java

baseserverthread.java

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

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.

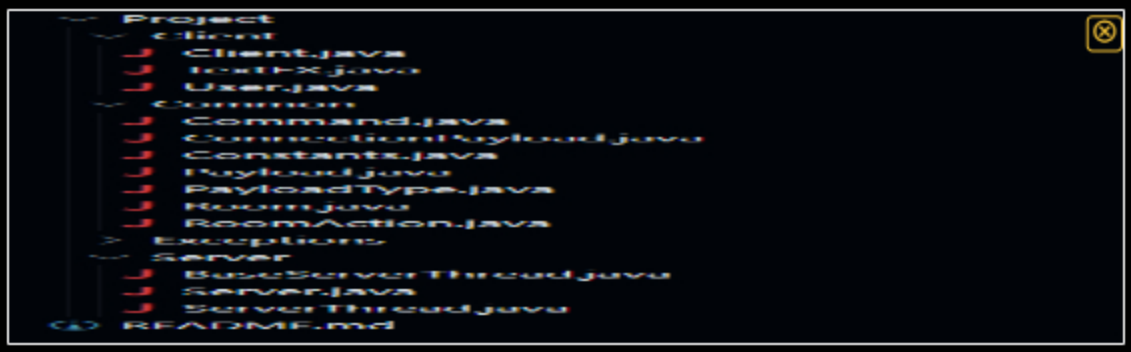
Saved: 7/14/2025 11:00:17 PM

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%



my workplace structure

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

Task #2 (0.25 pts.) - Github Details

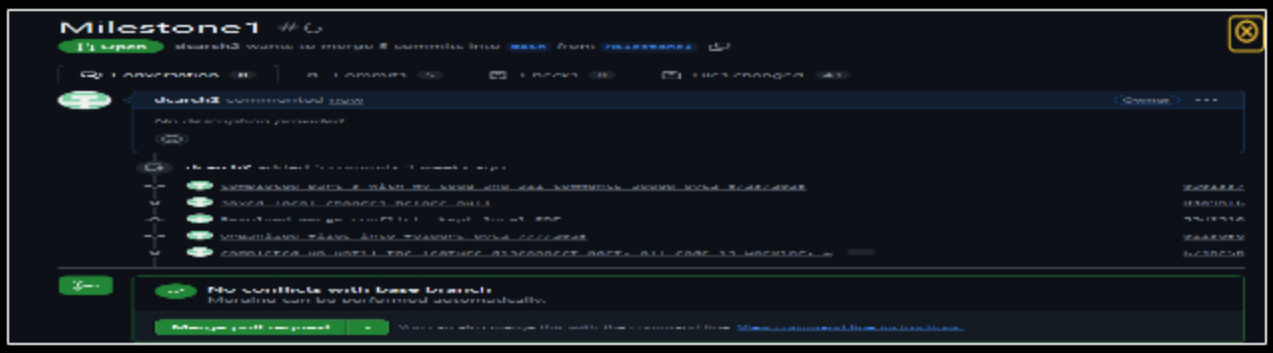
Progress: 100%

Part 1:

Progress: 100%

Details:

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



screenshot

Saved: 7/14/2025 11:07:31 PM

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-

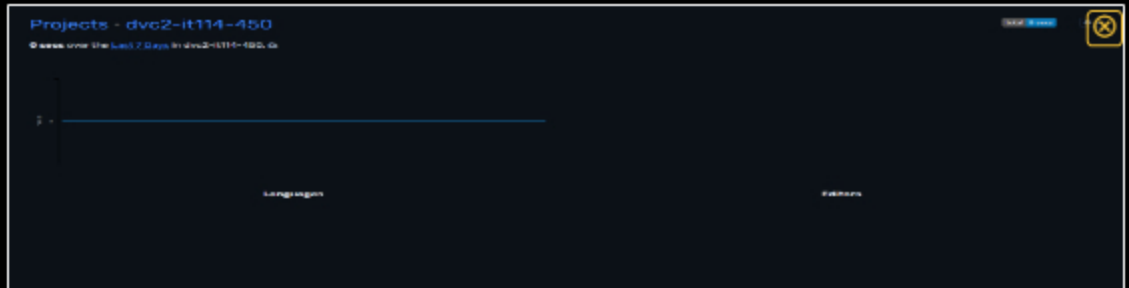
Saved: 7/14/2025 11:07:31 PM

Task #3 (0.25 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary



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