

Submission Worksheet

Submission Data

Course: IT114-003-F2025

Assignment: IT114 Module 4 Sockets Part3 Challenge

Student: Rayyan K. (rk975)

Status: Submitted | **Worksheet Progress:** 98%

Potential Grade: 9.95/10.00 (99.50%)

Received Grade: 0.00/10.00 (0.00%)

Started: 10/21/2025 7:24:50 PM

Updated: 10/21/2025 9:19:38 PM

Grading Link: <https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-module-4-sockets-part3-challenge/grading/rk975>

View Link: <https://learn.ethereallab.app/assignment/v3/IT114-003-F2025/it114-module-4-sockets-part3-challenge/view/rk975>

Instructions

• Overview Link: https://youtu.be/_029E_aBTFo

1. Ensure you read all instructions and objectives before starting.
2. Create a new branch from main called M4-Homework
 1. `git checkout main` (ensure proper starting branch)
 2. `git pull origin main` (ensure history is up to date)
 3. `git checkout -b M4-Homework` (create and switch to branch)
3. Copy the template code from here: [GitHub Repository - M4 Homework](#)
 - It includes Sockets Part1, Part2, and Part3. Put all into an M4 folder or similar if you don't have them yet (adjust package reference at the top if you chose a different folder name).
 - Make a copy of Part3 and call it Part3HW
 - Fix the package and import references at the top of each file in this new folder (Note: you'll only be editing files in Part3HW)
 - Immediately record to history
 - `git add .`
 - `git commit -m "adding M4 HW baseline files"`
 - `git push origin M4-Homework`
 - Create a Pull Request from M4-Homework to main and keep it open
4. Fill out the below worksheet
 - Each Problem requires the following as you work
 - Ensure there's a comment with your UCID, date, and brief summary of how the problem was solved
 - Code solution (add/commit periodically as needed)
 - Hint: Note how /reverse is handled
5. Once finished, click "Submit and Export"
6. 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 M4-Homework

4. On Github merge the pull request from M4-Homework to main

7. Upload the same PDF to Canvas

8. Sync Local

1. git checkout main

2. git pull origin main

Section #1: (3 pts.) Challenge 1 - Coin Flip

Progress: 100%

≡ Task #1 (3 pts.) - Implement a Coin Flip Command

Progress: 100%

Details:

- Client must capture the user entry and generate a valid command per the lesson details
 - Command format must be /flip
- ServerThread must receive the data and call the correct method on Server
- Server must expose a method for the logic and send the result to everyone
 - The message must be in the format of <who> flipped a coin and got <result> and be from the Server
- Add code to solve the problem (add/commit as needed)

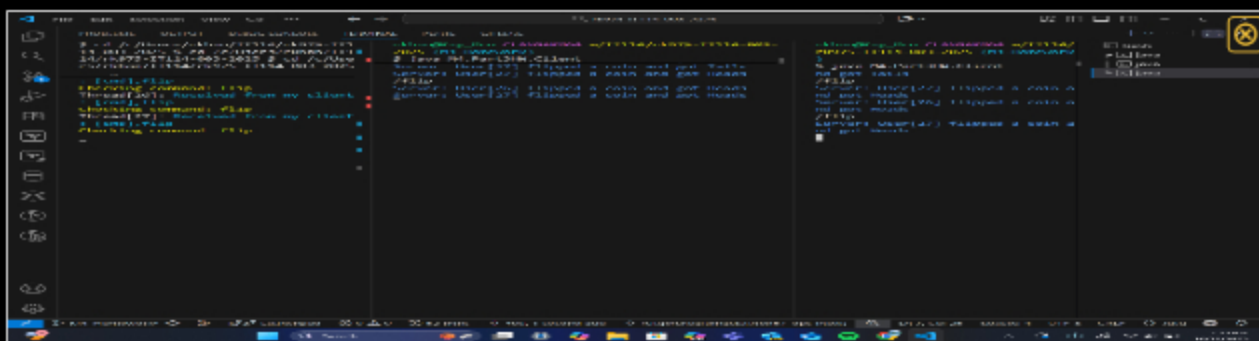
📁 Part 1:

Progress: 100%

Details:

Multiple screenshots are expected

1. Snippet of relevant code showing solution (with ucid/date comment) from Client
 - Should only need to edit processClientCommands()
2. Snippet of relevant code showing solution (with ucid/date comment) from ServerThread
 - Should only need to edit processCommand()
3. Snippet of relevant code showing solution (with ucid/date comment) from Server
 - Should only need to create a new method and pass the result message to relay()
4. Show 5 examples of the command being seen across all terminals (2+ Clients and 1 Server)
 1. This can be captured in one screenshot if you split the terminals side by side



IT114-1008/2025/

Homework/M4/Part3HW/Server.java

URL #3

<https://github.com/rayk101/rk975->

IT114-1008/2025/

Homework/M4/Part3HW/ServerThread.java



URL

<https://github.com/rayk101/rk975>



Saved: 10/21/2025 7:29:23 PM

⇒ Part 3:

Progress: 100%

Details:

Briefly explain how the code solves the challenge (note: this isn't the same as what the code does)

Your Response:

The code solves the challenge by adding a clear communication flow between client and server for a new /flip command. It correctly routes the request through each layer — the client captures the command, the server thread interprets it, and the server executes the logic and broadcasts the result without breaking existing functionality. This demonstrates understanding of distributed communication, command handling, and message broadcasting in a multi-client chat system.



Saved: 10/21/2025 7:29:23 PM

Section #2: (3 pts.) Challenge 2 - Private Message

Progress: 100%

≡ Task #1 (3 pts.) - Implement a Private Message Command

Progress: 100%

Details:

- Client must capture the user entry and generate a valid command per the lesson details
 - Command format must be /pm <target id> <message>
- ServerThread must receive the data and call the correct method on Server
- Server must expose a method for the logic
 - The message must be in the format of PM from <who>: <message> and be from the Server
 - The result must only be sent to the original sender and to the receiver/target
- Add code to solve the problem (add/commit as needed)

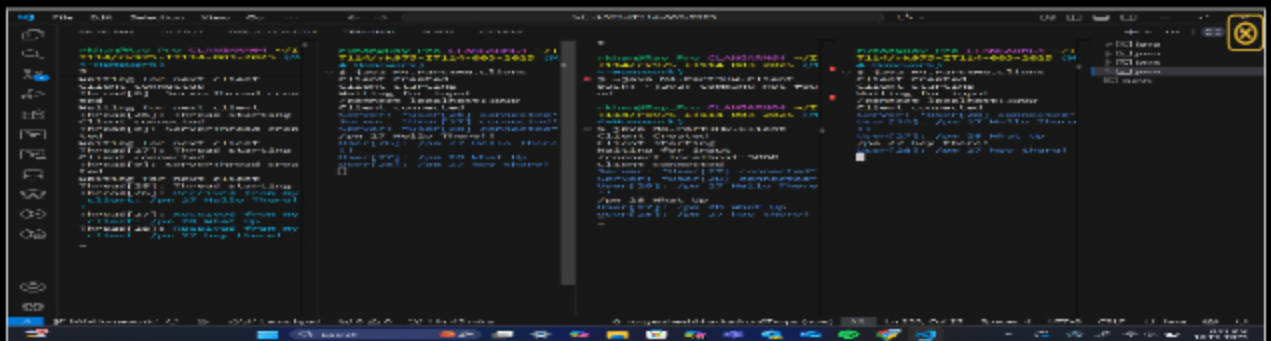
🖼 Part 1:

Progress: 100%

Details:

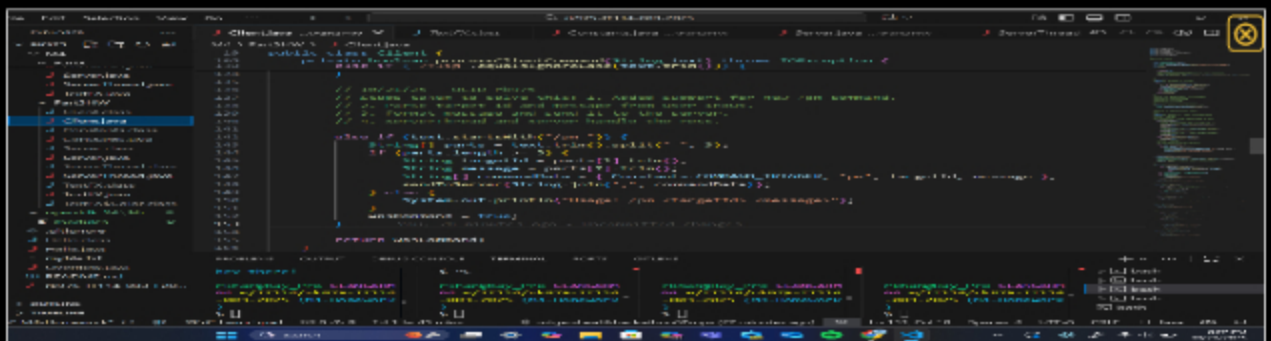
Multiple screenshots are expected

1. Snippet of relevant code showing solution (with ucid/date comment) from Client
 - Should only need to edit processClientCommands()
2. Snippet of relevant code showing solution (with ucid/date comment) from ServerThread
 - Should only need to edit processCommand()
3. Snippet of relevant code showing solution (with ucid/date comment) from Server
 - Should only need to create a new method and send the result message to just the sender and receiver
4. Show 3 examples of the command being seen across all terminals (3+ Clients and 1 Server)
 1. This can be captured in one screenshot if you split the terminals side by side
 2. Note: Only the sender and the receiver should see the private message (show variations across different users)



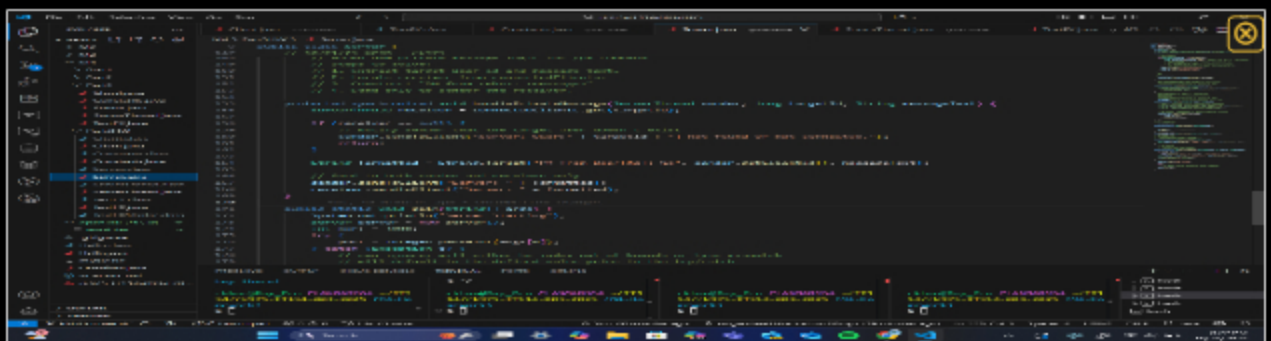
This screenshot shows an IDE with four terminal windows. The first window on the left shows code for the Client class, specifically the processClientCommands() method. The second window shows code for the ServerThread class, specifically the processCommand() method. The third window shows code for the Server class, specifically the processCommand() method. The fourth window on the right shows code for the Server class, specifically the processCommand() method. The code snippets are color-coded and include comments.

output



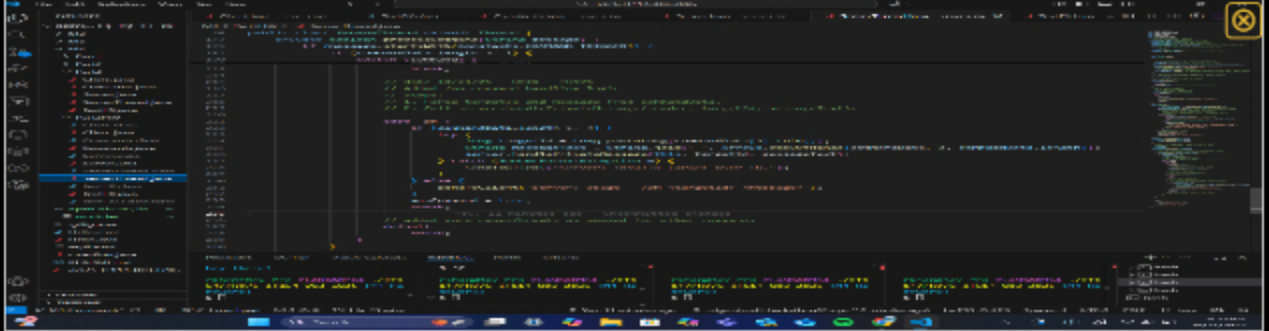
This screenshot shows an IDE with a single terminal window. The code snippets are color-coded and include comments. The code is for the Client class, specifically the processClientCommands() method. The code is for the ServerThread class, specifically the processCommand() method. The code is for the Server class, specifically the processCommand() method. The code snippets are color-coded and include comments.

client




This screenshot shows an IDE with a single terminal window. The code snippets are color-coded and include comments. The code is for the Client class, specifically the processClientCommands() method. The code is for the ServerThread class, specifically the processCommand() method. The code is for the Server class, specifically the processCommand() method. The code snippets are color-coded and include comments.

server



serverthread

 Saved: 10/21/2025 8:53:29 PM

Part 2:

Progress: 100%

Details:

Direct link to the file in the homework related branch from Github (should end in .java)

URL #1

<https://github.com/rayk101/rk975-IT114-100-2025/Homework/M4/Part3HW/Client.java>



URL

<https://github.com/rayk101/rk975>



URL #2

<https://github.com/rayk101/rk975-IT114-100-2025/Homework/M4/Part3HW/Server.java>



URL

<https://github.com/rayk101/rk975>



URL #3


<https://github.com/rayk101/rk975-IT114-100-2025/Homework/M4/Part3HW/ServerThread.java>



URL

<https://github.com/rayk101/rk975>



 Saved: 10/21/2025 8:53:29 PM

Part 3:

Progress: 100%

Details:

Briefly explain how the code solves the challenges (note: this isn't the same as what the code does)

Your Response:

The code solves the challenge by extending the existing socket-based client-server architecture to support targeted message routing through command-driven communication. It introduces a ~~new~~ command parsed at the client level, interpreted in each ServerThread, and dispatched via the server's handlePrivateMessage() method using synchronized access to shared client mappings. This approach enforces controlled message flow between threads, ensuring only the sender and

specified recipient receive the payload.



Saved: 10/21/2025 8:53:29 PM

Section #3: (3 pts.) Challenge 3 - Shuffle Message

Progress: 100%

≡ Task #1 (3 pts.) - Implement a Shuffle Message Command

Progress: 100%

Details:

- Client must capture the user entry and generate a valid command per the lesson details
 - Command format must be `/shuffle <message>`
- ServerThread must receive the data and call the correct method on Server
- Server must expose a method for the logic and send the result to everyone
 - The message must be in the format of `Shuffled from <who>: <shuffled_message>` and be from the Server
- Add code to solve the problem (add/commit as needed)

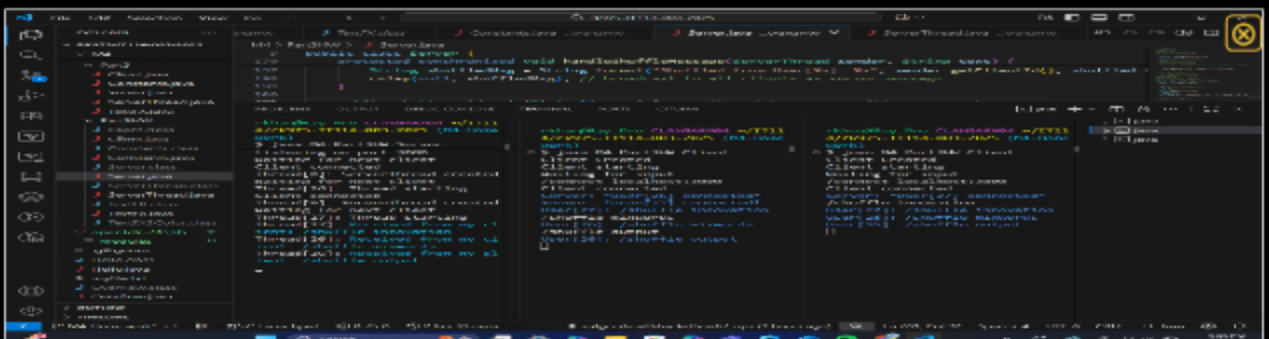
📁 Part 1:

Progress: 100%

Details:

Multiple screenshots are expected

1. Snippet of relevant code showing solution (with ucid/date comment) from Client
 - Should only need to edit `processClientCommands()`
2. Snippet of relevant code showing solution (with ucid/date comment) from ServerThread
 - Should only need to edit `processCommand()`
3. Snippet of relevant code showing solution (with ucid/date comment) from Server
 - Should only need to create a new method and do similar logic to `relay()`
4. Show 3 examples of the command being seen across all terminals (2+ Clients and 1 Server)
 1. This can be captured in one screenshot if you split the terminals side by side



output

```
File Edit View Window Help
M4 - Part3HW - Client.java
// M4 - Part3HW - Client.java
// Author: Rayk101
// Date: 10/21/2025
// Description: This program implements a client for a network-based game. It connects to a server and sends/receives data.

import java.io.*;
import java.net.*;
import java.util.*;

public class Client {
    private static final String SERVER_IP = "127.0.0.1";
    private static final int SERVER_PORT = 8080;
    private static final int BUFFER_SIZE = 1024;

    private Socket socket;
    private DataInputStream in;
    private DataOutputStream out;

    public Client() {
        try {
            socket = new Socket(SERVER_IP, SERVER_PORT);
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void start() {
        // Send initial command
        String command = "START";
        out.writeUTF(command);

        // Receive response
        String response = null;
        try {
            response = in.readUTF();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Print response
        System.out.println("Received: " + response);
    }

    public void stop() {
        // Close connection
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

server

```
M4 - Part3HW - Server.java
// M4 - Part3HW - Server.java
// Author: Rayk101
// Date: 10/21/2025
// Description: This program implements a server for a network-based game. It listens for client connections and processes commands.

import java.io.*;
import java.net.*;
import java.util.*;

public class Server {
    private static final int SERVER_PORT = 8080;
    private static final int BUFFER_SIZE = 1024;

    private ServerSocket serverSocket;
    private Socket socket;
    private DataInputStream in;
    private DataOutputStream out;

    public Server() {
        try {
            serverSocket = new ServerSocket(SERVER_PORT);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void start() {
        while (true) {
            try {
                socket = serverSocket.accept();
                in = new DataInputStream(socket.getInputStream());
                out = new DataOutputStream(socket.getOutputStream());

                // Receive command
                String command = in.readUTF();

                // Process command
                String response = null;
                if (command.equals("START")) {
                    response = "STARTED";
                } else {
                    response = "UNKNOWN";
                }

                // Send response
                out.writeUTF(response);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

client

```
M4 - Part3HW - ClientThread.java
// M4 - Part3HW - ClientThread.java
// Author: Rayk101
// Date: 10/21/2025
// Description: This program implements a client thread for a network-based game. It connects to a server and sends/receives data.

import java.io.*;
import java.net.*;
import java.util.*;

public class ClientThread extends Thread {
    private static final String SERVER_IP = "127.0.0.1";
    private static final int SERVER_PORT = 8080;
    private static final int BUFFER_SIZE = 1024;

    private Socket socket;
    private DataInputStream in;
    private DataOutputStream out;

    public ClientThread() {
        try {
            socket = new Socket(SERVER_IP, SERVER_PORT);
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        // Send initial command
        String command = "START";
        out.writeUTF(command);

        // Receive response
        String response = null;
        try {
            response = in.readUTF();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Print response
        System.out.println("Received: " + response);
    }
}
```

serverthread



Saved: 10/21/2025 9:12:29 PM

Part 2:

Progress: 100%

Details:

Direct link to the file in the homework related branch from Github (should end in .java)

URL #1

<https://github.com/rayk101/rk975-IT114-1006/2025/Homework/M4/Part3HW/Client.java>



URL

<https://github.com/rayk101/rk975>



URL #2

<https://github.com/rayk101/rk975-IT114-1006/2025/Homework/M4/Part3HW/Server.java>



URL

<https://github.com/rayk101/rk975>



<https://github.com/rayk101/rk975-IT114-1008/2025/Homework/M4/Part3HW/Server.java>

URL #3

[https://github.com/rayk101/rk975-](https://github.com/rayk101/rk975-IT114-1008/2025/Homework/M4/Part3HW/ServerThread.java)

[IT114-1008/2025/](https://github.com/rayk101/rk975-IT114-1008/2025/Homework/M4/Part3HW/ServerThread.java)

[Homework/M4/Part3HW/ServerThread.java](https://github.com/rayk101/rk975-IT114-1008/2025/Homework/M4/Part3HW/ServerThread.java)



URL

<https://github.com/rayk101/rk975>



Saved: 10/21/2025 9:12:29 PM

Part 3:

Progress: 100%

Details:

Briefly explain how the code solves the challenges (note: this isn't the same as what the code does)

Your Response:

The code solves the challenge by establishing a structured command-handling pipeline between the client, server thread, and server core. Each layer isolates responsibilities — the client detects and formats the /shuffle command, the ServerThread parses and routes it to the server, and the Server performs the shuffle logic and broadcasts the result. Thread-safe synchronization (synchronized) ensures consistent message delivery in a concurrent multi-client environment. This modular command architecture allows easy extension of new features like /flip, /pm, and /shuffle without breaking existing logic.



Saved: 10/21/2025 9:12:29 PM

Section #4: (1 pt.) Misc

Progress: 95%

Task #1 (0.33 pts.) - Github Details

Progress: 87%

Part 1:

Progress: 75%

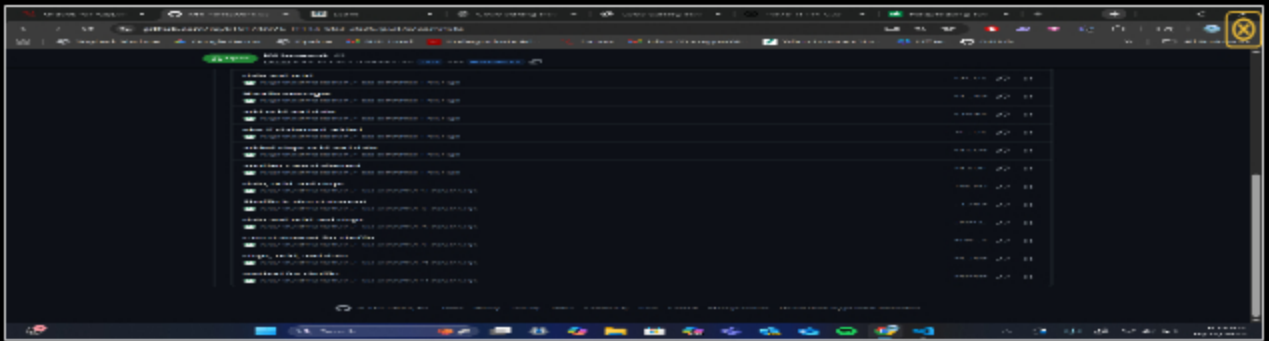
Details:

From the Commits tab of the Pull Request screenshot the commit history Following minimum should be present





PR



PR



Saved: 10/21/2025 9:15:34 PM

Part 2:

Progress: 100%

Details:

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

URL #1

<https://github.com/rayk101/rk975-IT114-0018-2025/>



URL

<https://github.com/rayk101/rk975>



Saved: 10/21/2025 9:15:34 PM

Task #2 (0.33 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





waketime



waketime



Saved: 10/21/2025 9:17:13 PM

Task #3 (0.33 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:

Overall, I learned how client-server communication works using Java sockets and threads. I understood how to design command-based interactions where clients send structured requests that the server interprets and processes. I also learned how synchronization and message relaying ensure data consistency and coordination across multiple connected clients in a concurrent environment.



Saved: 10/21/2025 9:18:31 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 was extending the existing command framework to include the /shuffle functionality. The project was already modular, so adding new logic required only small updates in Client, ServerThread, and Server without restructuring the system. Using existing methods like relay() and synchronized command handling made implementation and testing straightforward and efficient.



Saved: 10/21/2025 9:19:10 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 implementing and debugging the /shuffle command to ensure it worked correctly across multiple clients. It required careful handling of string manipulation, converting messages into character lists, and using Collections.shuffle() to randomize output while keeping thread safety.



Saved: 10/21/2025 9:19:38 PM