Submission Worksheet

Submission Data

Course: IT114-450-M2025
Assignment: IT114 Module 4 Sockets Part3 Challenge
Student: Colin R. (ctr26)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 6/21/2025 10:12:12 PM
Updated: 6/23/2025 7:30:55 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-module-4-sockets-part3-challenge/grading/ctr26
View Link: https://learn.ethereallab.app/assignment/v3/IT114-450-M2025/it114-module-4-sockets-part3-challenge/view/ctr26

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

```
124         } else if (text.startsWith(prefix:"/flip")) <- #118-124 else if (text.startsWith("/reverse"))
125         {
126             // ctr26 06-21-2025
127             // Added a new else if that checks for the flip command and sends the input to server thread
128             String[] commandData = { Constants.COMMAND_TRIGGER, "flip", text };
129             sendToServer(String.join(delimiter:",", commandData));
130             wasCommand = true;
131         } <- #125-131 else if (text.startsWith("/flip"))
132         return wasCommand;
```

```
133     } <- #101-133 private boolean processClientCommand(String text) throws IOEx...
```

Client code

```
205                 // ctr26 06-21-2025
206                 // Added a new case for the flip input sent from client
207                 // The case calls the handleFlip function in the server class where the rest of the logic is handled
208                 case "flip":
209                     server.handleFlip(this);
210                     wasCommand = true;
211                     break;
```

ServerThread code

```
134     // ctr26 06-21-2025
135     // Copied structure of the other command messages and added the custom logic to handle the coinflips
136     protected synchronized void handleFlip(ServerThread sender)
137     {
138         String result = "tails";
139         if ((int) (Math.random() * 2) == 1) result = "heads";        You, 26 minutes ago • Uncommitted changes
140         relay(sender, String.format(format:"Client[%d] flipped a coin and got %s", sender.getClientId(), result));
141     } <- #137-141 protected synchronized void handleFlip(ServerThread sender)
```

Server code

```
PROBLEMS   OUTPUT   TERMINAL   GITLENS   PORTS   DEBUG CONSOLE

User[31]: Client[31] flipped a coin and got tails
/fli[
User[31]: /fli[
/flip
User[31]: Client[31] flipped a coin and got heads
/flip
User[31]: Client[31] flipped a coin and got heads
/flip
User[31]: Client[31] flipped a coin and got tails
User[32]: Client[32] flipped a coin and got heads
User[32]: Client[32] flipped a coin and got tails

User[31]: Client[31] flipped a coin and got tails
User[31]: Client[31] flipped a coin and got tails
User[31]: /fli[
User[31]: Client[31] flipped a coin and got heads
User[31]: Client[31] flipped a coin and got heads
User[31]: Client[31] flipped a coin and got tails
/flip
User[32]: Client[32] flipped a coin and got heads
/flip
User[32]: Client[32] flipped a coin and got tails

Checking command: flip
Thread[31]: Received from my client: /fli[
Thread[31]: Received from my client: [cmd],flip,/flip
Checking command: flip
Thread[31]: Received from my client: [cmd],flip,/flip
Checking command: flip
Thread[31]: Received from my client: [cmd],flip,/flip
Checking command: flip
Thread[32]: Received from my client: [cmd],flip,/flip
Checking command: flip
Thread[32]: Received from my client: [cmd],flip,/flip
Checking command: flip
```

Output

Saved: 6/21/2025 10:18:58 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/ColinRafferty7/ctr26-IT114-450M4-Homework/Module4/Part3/Client.java

URL
https://github.com/ColinRafferty7

URL #2
https://github.com/ColinRafferty7/ctr26-IT114-450-M4-Homework/Module4/Part3/Server.java

https://github.com/ColinRafferty7

URL #3
https://github.com/ColinRafferty7/ctr26-IT114-450-M4-Homework/Module4/Part3/ServerThread.java

https://github.com/ColinRafferty7

Saved: 6/21/2025 10:18:58 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:

I followed the same structure that the default commands had. First take the user input from command side and converting it into readable data for the thread file, which then triggers the corresponding command in the server file. Finally, the server handles the flipping logic and relays the data back to all clients.

Saved: 6/21/2025 10:18:58 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 pass the result message to `relay()`
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)

```
} else if (text.startsWith(prefix:"/pm")) <- #125-131 else if (text.startsWith("/flip"))
{
    // ctr26 06-21-2025
    // Added a new else if statement for the pm command and sends the input to server thread
    String[] commandData = { Constants.COMMAND_TRIGGER, "pm", text };
    sendToServer(String.join(delimiter:",", commandData));
    wasCommand = true;
} <- #132-138 else if (text.startsWith("/pm"))
```

Client code

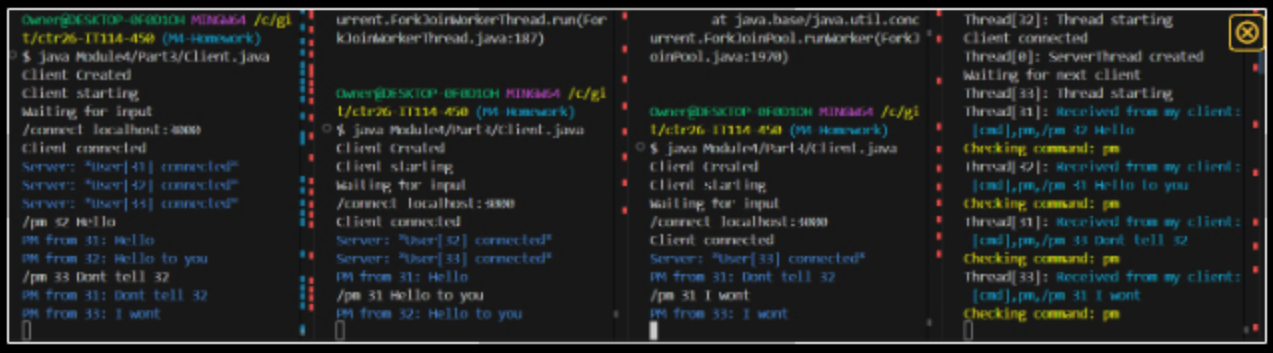
```
// ctr26 06-23-2025
// Added a new case that splits the command data into an array of the entire user input
// The receivers client id is then extracted and the rest of the data is kept as the message string and sent to the server
case "pm":
    String[] text = String.join(delimiter:" ", Arrays.copyOfRange(commandData, from:2, commandData.length)).split(regex:" ");
    String pMessage = String.join(delimiter:" ", Arrays.copyOfRange(text, from:2, text.length));
    long receiverId = Long.parseLong(text[1]);
    server.handlePM(this, receiverId, pMessage);
    wasCommand = true;
    break;
```

ServerThread code

```
// ctr26 06-23-2025
// Added a new method that takes the entered thread id and gets the serverthread value from the hash map
// The final message is constructed and sent out to both the sending client and receiving client
protected synchronized void handlePM(ServerThread sender, long receiverId, String message)
{
    ServerThread receiver = connectedClients.get(receiverId);
    String finalMessage = String.format(format:"PM from %d: %s", sender.getClientId(), message);
    receiver.sendToClient(finalMessage);
    sender.sendToClient(finalMessage);
}
```

Server code

Server code



Ouput

Saved: 6/23/2025 6:28:10 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/ColinRafferty7/ctr26-IT114-450/M4-Homework/Module4/Part3

URL
https://github.com/ColinRafferty7

Saved: 6/23/2025 6:28:10 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 problem was solved by utilizing the hash map with all of the connect clients. By converting the user input into a client id, I can pull the corresponding server thread from the hash map and use it to directly send the message.

Saved: 6/23/2025 6:28:10 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

```java
} else if (text.startsWith(prefix:"/shuffle")) <- #132-138 else if (text.startsWith("/pm"))
{
    // ctr26 06-23-2025
    // Added a new else if statement for the shuffle command and sends the input to server thread
    String[] commandData = { Constants.COMMAND_TRIGGER, "shuffle", text };
    sendToServer(String.join(delimiter:",", commandData));
    wasCommand = true;
} <- #139-145 else if (text.startsWith("/shuffle"))
```

Client code

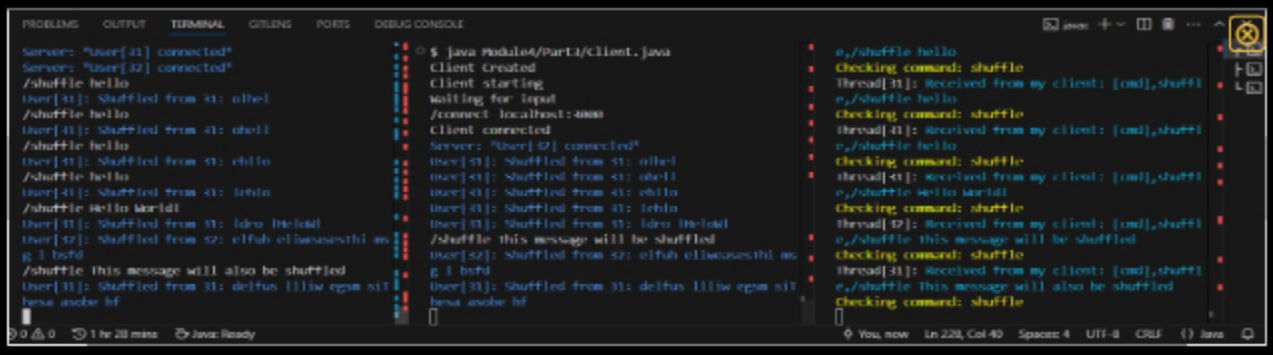
```java
// ctr26 06-23-2025
// Added new case that takes the input and shuffles it
// the shuffle function goes through each character in the input and adds it onto either the start or end of final message
case "shuffle":        You, 16 minutes ago • Uncommitted changes
    String[] shuffleInput = String.join(delimiter:" ", Arrays.copyOfRange(commandData, from:2, commandData.length)).split(regex:" ");
    String[] temp = (String.join(delimiter:" ", Arrays.copyOfRange(shuffleInput, from:1, shuffleInput.length))).split(regex:"");
    String shuffledMessage = "";
    for (String character : temp)
    {
        if ((int) (Math.random() * 2) == 1)
        {
            shuffledMessage = shuffledMessage + character;
        }
        else
        {
            shuffledMessage = character + shuffledMessage;
        }
    } <- #227-230 for (String character : temp)
```

```
server.handleShuffle(this, shuffledMessage);
wasCommand = true;
break;
```

ServerThread code

```
// ctr26 06-23-2025
// Added new method that takes the shuffled string and relaysys it in the desired format
protected synchronized void handleShuffle(ServerThread sender, String message)
{
    String finalMessage = String.format(format:"Shuffled from %d: %s", sender.getClientId(), message);
    relay(sender, finalMessage);
}
```

Server code



Output

Saved: 6/23/2025 7:21:36 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/ColinRafferty7/ctr26-IT114-450-M4-Homework/Module4/Part3

URL
https://github.com/ColinRafferty7

Saved: 6/23/2025 7:21:36 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 problem by seperating the message part of the input and splitting it into a list of its characters. Then, a for loop is used to take each letter individually and either add it onto the start or end of a result string, dependent on a random function.

Saved: 6/23/2025 7:21:36 PM

Section #4: (1 pt.) Misc

Progress: 100%

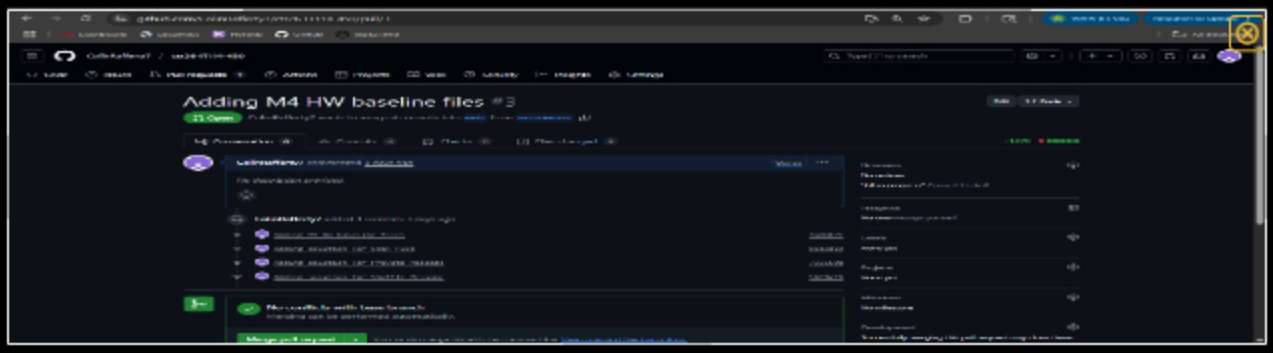
Task #1 (0.33 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



Pull request

Saved: 6/23/2025 7:24:01 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/ColinRafferty7/ctr26-IT114-453/

URL
https://github.com/ColinRafferty7

Saved: 6/23/2025 7:24:01 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

Wakatime top

Wakatime bottom

Saved: 6/23/2025 7:25:09 PM

Task #3 (0.33 pts.) - Reflection

Progress: 0%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

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

Your Response:

For this assignment, I learned how connections work through code. I have never worked with multiple programs running simultaneously and having them communicate with each other. It was a pretty good exercise that really showed me the steps that each program takes in order to complete a simple connection like a message.

Saved: 6/23/2025 7:26:40 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 this assignment was the taking the client input. After looking through the code a bit, I realized that the structure was already layed out for me, and I just needed to copy the same format that the default commands used. Then, I was able to use that logic for each problem.

Saved: 6/23/2025 7:28:15 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 this assignment was figuring out how to send the message to only specific clients. The problem specified that we should use the relay function to send out the final message, but that function only ever output the message to all users in the hashmap. So, the only ways that I figured I could solve the problem was to fiddle with the hashmap in order to specify the targets, or what I settled on was to just forgo the relay command. That made the problem make a lot more sense to me becaus I was ble to pull the exact target from the has map using the client id, and then use the send to client function to send the message.

Saved: 6/23/2025 7:30:55 PM