# Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-006-S2024/it114-sockets-part-1-3-checkpoint/grade/ms75

IT114-006-S2024 - [IT114] Sockets Part 1-3-Checkpoint

Submissions:

Submission Selection

1 Submission [active] 2/19/2024 1:48:56 PM

## Instructions

COLLAPSE

- Create a new branch for this assignment
- Go through the socket lessons and get each part implemented (parts 1-3)
  - You'll probably want to put them into their own separate folders/packages (i.e., Part1, Part2, Part3) These are for your reference
  - Part 3, below, is what's necessary for this HW
  - https://github.com/MattToegel/IT114/tree/Module4/Module4/Part3
- Create a new folder called Part3HW (copy of Part3)
- Make sure you have all the necessary files from Part3 copied here and fix the package references at the top of each file
  - Add/commit/push the branch
  - Create a pull request to main and keep it open
- Implement **two** of the following **server-side** activities for all connected clients (majority of the logic should be processed server-side and broadcasted/sent to all clients if/when applicable)
  - Simple number guesser where all clients can attempt to guess while the game is active
    - Have a /start command that activates the game allowing guesses to be interpreted
    - Have a /stop command that deactivates the game, guesses will be treated as regular messages (i.e., guess messages are ignored)
    - Have a guess command that include a value that is processed to see if it matches the hidden number (i.e., */guess 5*)
      - Guess should only be considered when the game is active
      - The response should include who guessed, what they guessed, and whether or not it was correct (i.e., Bob guessed 5 but it was not correct)
    - No need to implement complexities like strikes
  - Coin toss command (random heads or tails)
    - Command should be something logical like /flip or /toss or /coin or similar
    - The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)
  - Dice roller given a command and text format of "/roll #d#" (i.e., roll 2d6)
    - Command should be in the format of /roll #d# (i.e., roll 1d10)
    - The result should mention *who* did *what* and got what *result* (i.e., Bob rolled 1d10 and got 7)
  - Math game (server outputs a basic equation, first person to guess it correctly gets congratulated and a new equation is given)
    - Have a /start command that activates the game allowing equaiton to be answered
    - Have a /stop command that deactivates the game, answers will be treated as regular messages (i.e., any game related commands when stopped will be ignored)
    - Have an answer command that include a value that is processed to see if it matches the hidden number (i.e., */answer 15*)

The response should include who answered, what they answered, and whether or not it was correct (i.e., Bob answered 5 but it was not correct)
- Private message (a client can send a message targetting another client where only the two can see the messages)
  - Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
  - The server should properly check the target audience and send the response to the original sender and to the receiver (no one else should get the message)
  - Alternatively (make note if you do this and show evidence) you can add support to private message multiple people at once. Evidence should show a larger number of clients than the target list of the private message to show it works. Note to grader: if this is accomplished add 0.5 to total final grade on Canvas
- Message shuffler (randomizes the order of the characters of the given message)
  - Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
  - The message should be sent to all clients showing it's from the user but randomized
    - Example: Bob types /*command* hello and everyone recevies Bob: lleho

Fill in the below deliverables
Save the submission and generated output PDF
Add the PDF to the Part3HW folder (local)
Add/commit/push your changes
Merge the pull request
Upload the same PDF to Canvas

**Branch name:** M4-Sockets3-Homework

Tasks: 7 Points: 10.00

● **Baseline** (2 pts.)

COLLAPSE

● COLLAPSE

## Task #1 - Points: 1

**Text: Demonstrate Baseline Code Working**

Details:
This can be a single screenshot if everything fits, or can be multiple screenshots

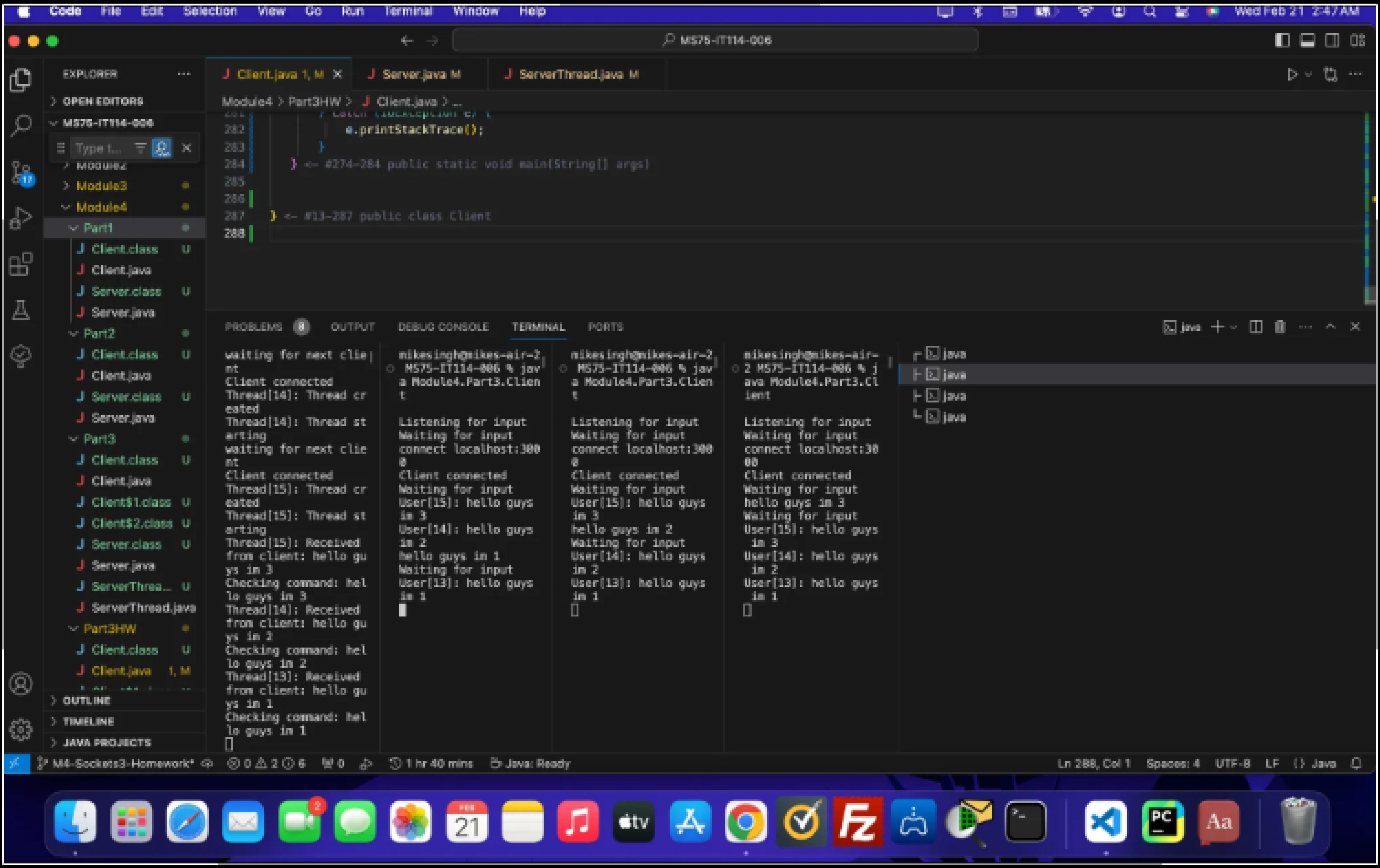
Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Server terminal/instance is clearly shown/noted |
| #2 | 1 | At least 3 client terminals should be visible and noted |
| #3 | 1 | Each client should correctly receive all broadcasted/shared messages |
| #4 | 1 | Captions clearly explain what each screenshot is showing |
| #5 | 1 | Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW |

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
            e.printStackTrace();
        }
    } <- #274-284 public static void main(String[] args)

} <- #13-287 public class Client
```

```
waiting for next clie
nt
Client connected
Thread[14]: Thread cr
eated
Thread[14]: Thread st
arting
waiting for next clie
nt
Client connected
Thread[15]: Thread cr
eated
Thread[15]: Thread st
arting
Thread[15]: Received
from client: hello gu
ys in 3
Checking command: hel
lo guys in 3
Thread[14]: Received
from client: hello gu
ys in 2
Checking command: hel
lo guys in 2
Thread[13]: Received
from client: hello gu
ys in 1
Checking command: hel
lo guys in 1
```

```
mikesingh@mikes-air-2
 MS75-IT114-006 % jav
a Module4.Part3.Clien
t

Listening for input
Waiting for input
connect localhost:300
0
Client connected
Waiting for input
User[15]: hello guys
in 3
User[14]: hello guys
in 2
hello guys in 1
Waiting for input
User[13]: hello guys
in 1
```

```
mikesingh@mikes-air-2
 MS75-IT114-006 % jav
a Module4.Part3.Clien
t

Listening for input
Waiting for input
connect localhost:300
0
Client connected
Waiting for input
User[15]: hello guys
in 3
hello guys in 2
Waiting for input
User[14]: hello guys
in 2
User[13]: hello guys
in 1
```

```
mikesingh@mikes-air-
2 MS75-IT114-006 % j
ava Module4.Part3.Cl
ient

Listening for input
Waiting for input
connect localhost:30
00
Client connected
Waiting for input
hello guys in 3
Waiting for input
User[15]: hello guys
 in 3
User[14]: hello guys
 in 2
User[13]: hello guys
 in 1
```

Screenshot shows my 4 open terminals with the server, client1 2 and 3(left to right). On the left hand side my module4 directory is open showing all the files for parts 1-3 as well as the compiled files. On each client terminal I typed Hello guys followed by the client number(1-3) and all clients were broadcasted the messages.

Checklist Items (0)

● Feature 1 (3 pts.)

^COLLAPSE^

●

^COLLAPSE^

## Task #1 - Points: 1

**Text: What feature did you pick? Briefly explain how you implemented it**

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| #1 | 1 | Feature is clearly stated (best to copy/paste it from above) |
| #2 | 1 | Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task) |

Response:

The first feature I chose was the

Coin toss command (random heads or tails)

> Command should be something logical like /flip or /toss or /coin or similar
> The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)

I implemented this feature by first adding the lines :

```
}else if (text.equalsIgnoreCase("Play Coin Toss")) {
        playerName = Name();
        coinToss();
        return true;
```

These lines were added into the processCommand boolean to check for the command Play Coin Toss which is the command I set to call the coinToss() method. The coinToss method contains a which is randomly generated and called where heads is true and tails is false. The client name and result is then printed.

COLLAPSE

## Task #2 - Points: 1

**Text: Add screenshot(s) showing the implemented feature working (code and output)**

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

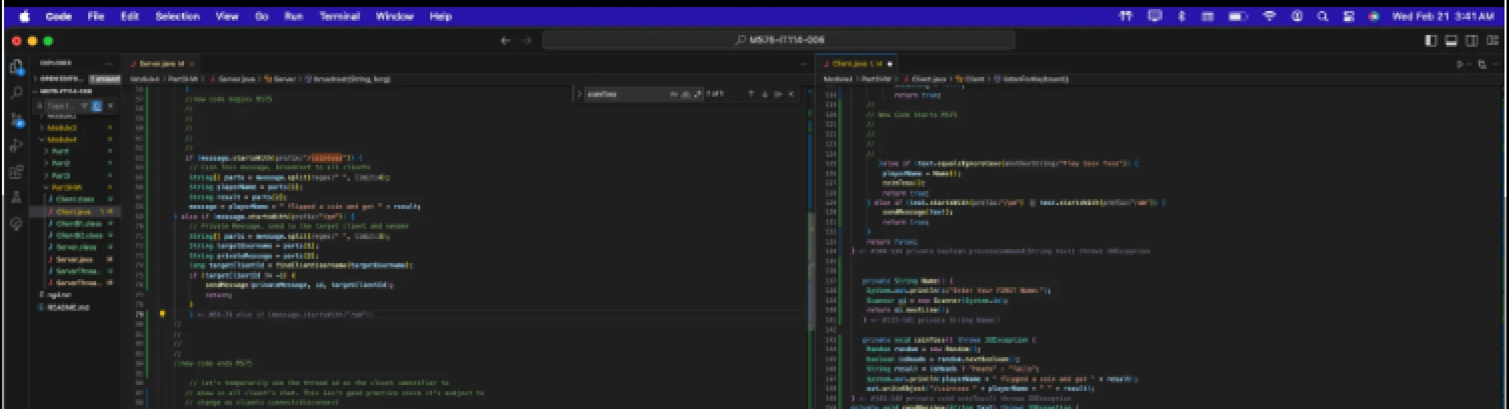
Checklist *The checkboxes are for your own tracking

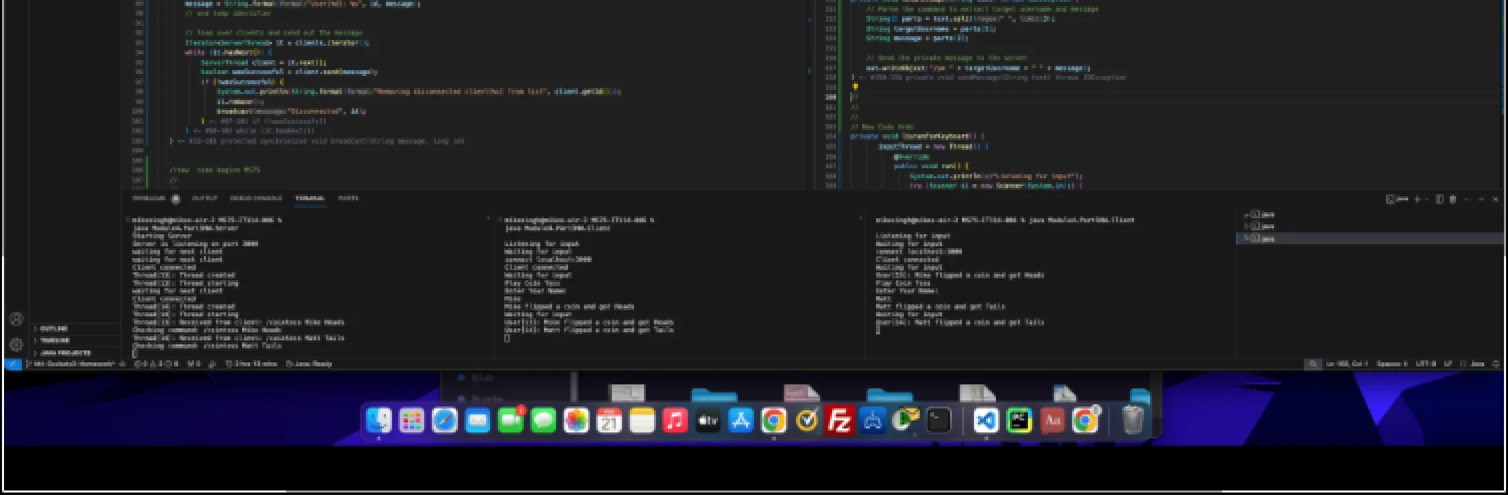
| # | Points | Details |
|---|---|---|
| #1 | 1 | Output is clearly shown and captioned |
| #2 | 1 | Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code. |

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Screenshot shows 2 Clients connected to the server and a compiled server terminal running. Both clients call the "Play Coin Test" command and all clients get the message saying what player got which face of the coin. The top left shows most of the changes for the Server.java file and the top right shows most of the changes for the Client.java file for the Coin Toss implementation.

Checklist Items (0)

● Feature 2 (3 pts.)

^COLLAPSE^

● ^COLLAPSE^

### Task #1 - Points: 1

**Text: What feature did you pick? Briefly explain how you implemented it**

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| ☐ #1 | 1 | Feature is clearly stated (best to copy/paste it from above) |
| ☐ #2 | 1 | Explanation sufficiently and concisely describes implementation (should be aligned with code snippets in related task) |

Response:

The second feature I chose was

- Private message (a client can send a message targetting another client where only the two can see the messages)
  - Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
  - The server should properly check the target audience and send the response to the original sender and to the receiver (no one else should get the message)

I first added the code:

```
} else if (text.startsWith("/pm") || text.startsWith("/dm")) {
        sendMessage(text);
        return true;
    }
```

into the Client.java file in the processCommand boolean to add the command line /pm and /dm. If the command is found then the sendMessage method is called.

found then the sendMessage method is called.

Then in the Server.java file most of the code was added at lines 106 - 136 which takes the Users name. I also added the clients name variable into the serverthreads.java run method to have the clients name be asked for once connected.
In the Server.java file I had to add a few methods for getting the client id and only sending the message to that id. I also added a sendMessage method with a sender and target recipient.

**Task #2 - Points: 1**

**Text: Add screenshot(s) showing the implemented feature working (code and output)**

Details:
Add screenshots of the relevant code changes AND relevant output during runtime

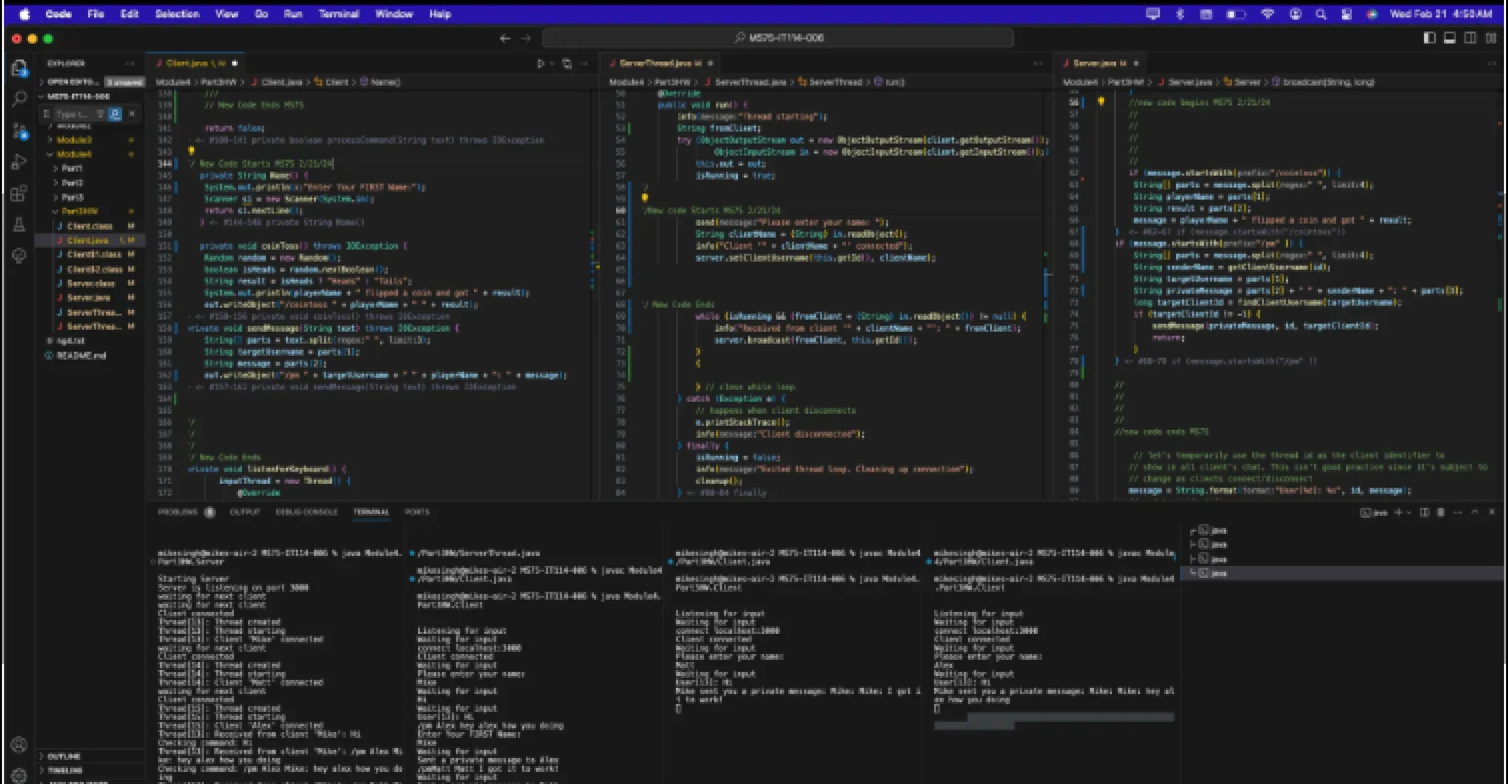
Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| ☐ #1 | 1 | Output is clearly shown and captioned |
| ☐ #2 | 1 | Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code. |

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```java
        ///
        // New Code Ends MS75

        return false;
    } <- #108-141 private boolean processCommand(String text) throws IOException

// New Code Starts MS75 2/21/24
    private String Name() {
        System.out.println(x:"Enter Your FIRST Name:");
        Scanner si = new Scanner(System.in);
        return si.nextLine();
    } <- #144-148 private String Name()

    private void coinToss() throws IOException {
        Random random = new Random();
        boolean isHeads = random.nextBoolean();
        String result = isHeads ? "Heads" : "Tails";
        System.out.println(playerName + " flipped a coin and got " + result);
        out.writeObject("/cointoss " + playerName + " " + result);
    } <- #150-156 private void coinToss() throws IOException
private void sendMessage(String text) throws IOException {
        String[] parts = text.split(regex:" ", limit:3);
        String targetUsername = parts[1];
        String message = parts[2];
        out.writeObject("/pm " + targetUsername + " " + playerName + ": " + message);
    } <- #157-162 private void sendMessage(String text) throws IOException



//
//
//
// New Code Ends
private void listenForKeyboard() {
        inputThread = new Thread() {
            @Override
```

```java
    @Override
    public void run() {
        info(message:"Thread starting");
        String fromClient;
        try (ObjectOutputStream out = new ObjectOutputStream(client.getOutputStream());
                ObjectInputStream in = new ObjectInputStream(client.getInputStream());) {
            this.out = out;
            isRunning = true;
//

//New code Starts MS75 2/21/24
            send(message:"Please enter your name: ");
            String clientName = (String) in.readObject();
            info("Client '" + clientName + "' connected");
            server.setClientUsername(this.getId(), clientName);


// New Code Ends
            while (isRunning && (fromClient = (String) in.readObject()) != null) {
                info("Received from client '" + clientName + "': " + fromClient);
                server.broadcast(fromClient, this.getId());
            }
            {

            } // close while loop
        } catch (Exception e) {
            // happens when client disconnects
            e.printStackTrace();
            info(message:"Client disconnected");
        } finally {
            isRunning = false;
            info(message:"Exited thread loop. Cleaning up connection");
            cleanup();
        } <- #80-84 finally
```

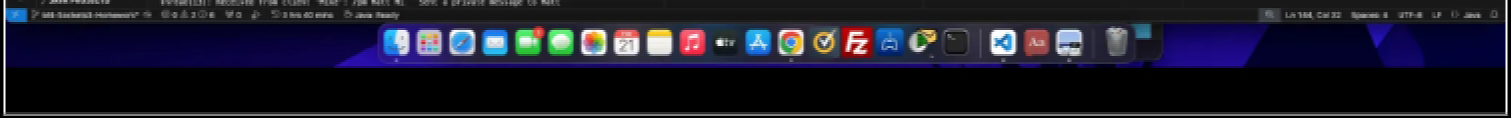
```java
        //new code begins MS75 2/21/24
        //
        //
        //
        //
        //
        if (message.startsWith(prefix:"/cointoss")) {
         String[] parts = message.split(regex:" ", limit:4);
         String playerName = parts[1];
         String result = parts[2];
         message = playerName + " flipped a coin and got " + result;
    } <- #62-67 if (message.startsWith("/cointoss"))
    if (message.startsWith(prefix:"/pm" )) {
         String[] parts = message.split(regex:" ", limit:4);
         String senderName = getClientUsername(id);
         String targetUsername = parts[1];
         String privateMessage = parts[2] + " " + senderName + ": " + parts[3];
         long targetClientId = findClientUsername(targetUsername);
         if (targetClientId != -1) {
             sendMessage(privateMessage, id, targetClientId);
             return;
         }
    } <- #68-78 if (message.startsWith("/pm" ))

    //
    //
    //
    //
    //new code ends MS75

         // let's temporarily use the thread id as the client identifier to
        // show in all client's chat. This isn't good practice since it's subject to
        // change as clients connect/disconnect
        message = String.format(format:"User[%d]: %s", id, message);
```

```
mikesingh@mikes-air-2 MS75-IT114-006 % java Module4.Part3HW.Server

Starting Server
Server is listening on port 3000
waiting for next client
waiting for next client
Client connected
Thread[13]: Thread created
Thread[13]: Thread starting
Thread[13]: Client 'Mike' connected
waiting for next client
Client connected
Thread[14]: Thread created
Thread[14]: Thread starting
Thread[14]: Client 'Matt' connected
waiting for next client
Client connected
Thread[15]: Thread created
Thread[15]: Thread starting
Thread[15]: Client 'Alex' connected
Thread[13]: Received from client 'Mike': Hi
Checking command: Hi
Thread[13]: Received from client 'Mike': /pm Alex Mike: hey alex how you doing
Checking command: /pm Alex Mike: hey alex how you doing
```

```
./Part3HW/ServerThread.java

mikesingh@mikes-air-2 MS75-IT114-006 % javac Module4/Part3HW/Client.java

mikesingh@mikes-air-2 MS75-IT114-006 % java Module4.Part3HW.Client

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
Please enter your name:
Mike
Waiting for input
Hi
Waiting for input
User[13]: Hi
/pm Alex hey alex how you doing
Enter Your FIRST Name:
Mike
Waiting for input
Sent a private message to Alex
/pmMatt Matt I got it to work!
Waiting for input
```

```
mikesingh@mikes-air-2 MS75-IT114-006 % javac Module4/Part3HW/Client.java

mikesingh@mikes-air-2 MS75-IT114-006 % java Module4.Part3HW.Client

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
Please enter your name:
Matt
Waiting for input
User[13]: Hi
Mike sent you a private message: Mike: Mike: I got it to work!
```

```
mikesingh@mikes-air-2 MS75-IT114-006 % javac Module4/Part3HW/Client.java

mikesingh@mikes-air-2 MS75-IT114-006 % java Module4.Part3HW.Client

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
Please enter your name:
Alex
Waiting for input
User[13]: Hi
Mike sent you a private message: Mike: Mike: hey alex how you doing
```



This screenshot shows my client.java file with changes, mainly showing the addition of the sendMessage method. There were a few changes i made to the ServerThread.java file to the run method so when a client connects they will be prompted for their name. Then the changes on the Server.java file were shown finding the correct targeted receiver of the private message. The 4 terminals show the server being connected then a Client named Mike, client named Matt and a Client named Alex. Mike Send private messages to Matt and Alex and only they received them.

Checklist Items (0)

● **Misc** (2 pts.)

^COLLAPSE^

● ^COLLAPSE^

## Task #1 - Points: 1

**Text: Reflection: Did you have an issues and how did you resolve them? If no issues, what did you learn during this assignment that you found interesting?**

Checklist *The checkboxes are for your own tracking

| # | Points | Details |
|---|---|---|
| ☐ #1 | 1 | An issue or learning is clearly stated |
| ☐ #2 | 1 | Response is a few reasonable sentences |

Response:

I had alot of issues trying to get the private message feature working. At first when i typed /pm and the clients name, the only way I could get it to run was if I went on all the clients first and flipped a coin and got their name from there first. If i just connected all the clients i had no way of setting  a name for them. I got a little confused as I was trying to do both features at once. I realized it would have been easier doing the private Message feature first and then the flip a coin one. I also got a better understanding of server threads and linking client data to a thread in between 2 clients.

● ^COLLAPSE^

## Task #2 - Points: 1

**Text: Pull request link**

Details:
URL should end with /pull/# and be related to this assignment

URL #1
https://github.com/mikes1302/MS75-IT114-006/pull/7

End of Assignment