Submission Worksheet

CLICK TO GRADE

https://learn.etherealab.app/assignment/IT114-006-S2024/it114-sockets-part-1-3-checkpoint/grade/sb57

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

Submissions:

Submission Selection

1 Submission [active] 4/30/2024 5:06:54 AM

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
5. 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
6. Have a /start command that activates the game allowing guesses to be interpreted
1. Have a /stop command that deactivates the game, guesses will be treated as regular messages (i.e., guess messages are ignored)
2. 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 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

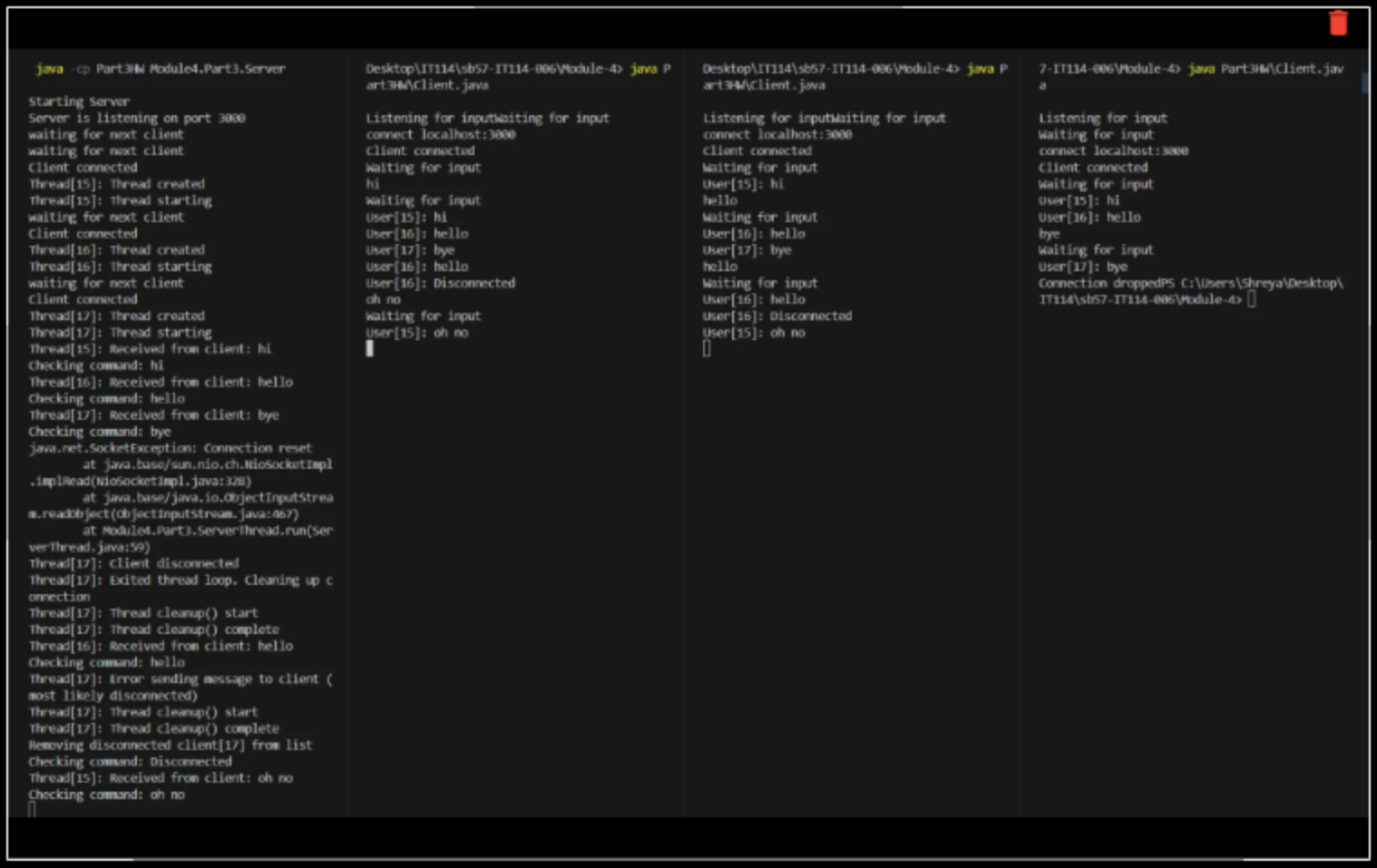
#	Points	Details
#1	1	Server terminal/instance is clearly shown/noted
#2	1	At least 3 client terminals should be visible and noted
#3	1	Each client should correctly receive all broadcasted/shared messages
	1	Captions clearly explain what each screenshot is showing

#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



This is a screenshot of me using the baseline code on the terminal. It shows 3 clients connected to one server at the same time receiving all broadcasts and messages. It also shows how the connection handles if one client disconnects.

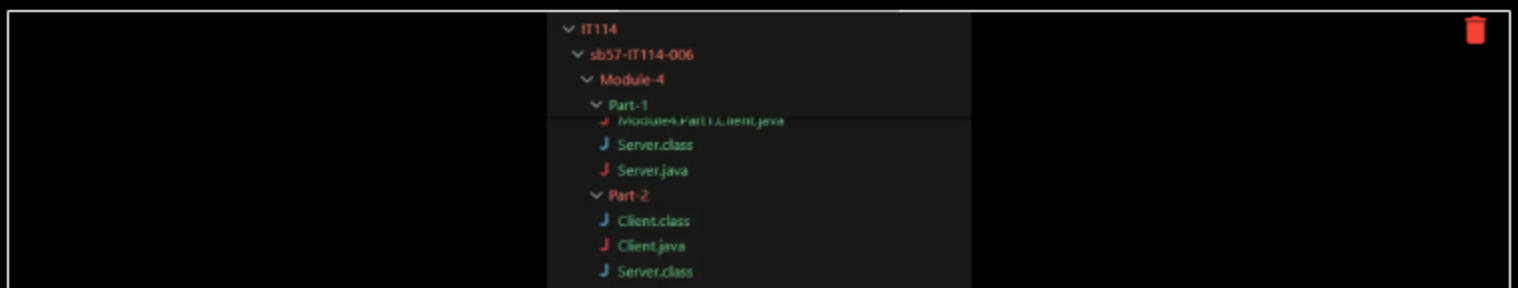
Checklist Items (4)

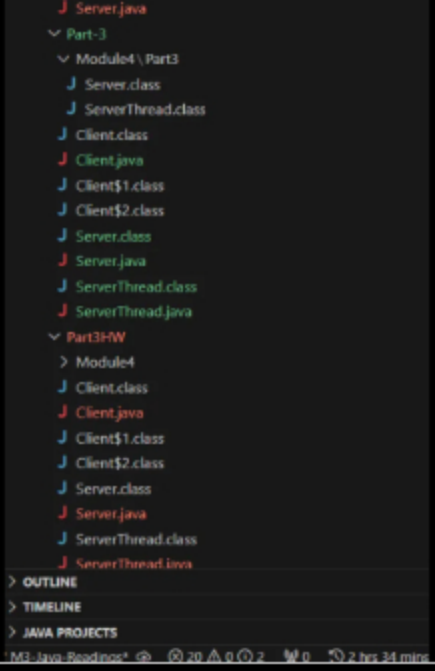
#1 Server terminal/instance is clearly shown/noted

#2 At least 3 client terminals should be visible and noted

#3 Each client should correctly receive all broadcasted/shared messages

#4 Captions clearly explain what each screenshot is showing





This screenshot shows on my vscode panel that I have correctly grabbed and stored Parts 1-3 and have them in my repository alongside Part3HW.

Checklist Items (2)

#4 Captions clearly explain what each screenshot is showing

#5 Include a screenshot showing you grabbed Parts 1-3 correctly and have them in your repository alongside Part3HW

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:

I chose the Number Guessing Game feature to implement:

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

I added both a start and stop command which activates and deactivates the game, and a guess command that determines if the correct number was guessed. Guesses are only considered to be directed to the game when it is active. Lastly, as my screenshot below shows, the server mentions which client the guess came from.

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

```
10  public class Server {
11      int port = 3001;
12      // connected clients
13      private List<ServerThread> clients = new ArrayList<ServerThread>();
14
15      //initializing variables for number guesser game implementation
16      private boolean gameActive = false;          //Shreya Bose
17      private int secretNumber;                    //sb57
18                                                   //February 19,2024
19      private void start(int port) {
```

```
20          this.port = port;
```

Screenshot shows initializing variables for the game implementation.

Checklist Items (1)

#2 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.

ServerThread.java 5 | Server.java 9+ | Client.java 1

IT114 > sb57-IT114-006 > Module-4 > Part3HW > Server.java > Server > processCommand(String, long)

```
11  public class Server {
75      private boolean processCommand(String message, long clientId){
88              return true;
89          }
90
91          //Number Guessing Game
92          if (gameActive) {
93              if (message.startsWith(prefix:"guess: ")) {
94                  try {                                                   //Shreya Bose
95                      int guess = Integer.parseInt(message.split(regex:" ")[1]);          //sb57
96                      boolean isCorrect = (guess == secretNumber);        //February 19, 2024
97                      String response = isCorrect ? "correct" : "incorrect";
98                      broadcast(String.format(format:"User[%d] guessed %d and it was %s", clientId, guess, response), clientId);
99                      if (isCorrect) {
100                         gameActive = false;
101                     }
102                     return true;
103                 } catch (NumberFormatException e) {
104                     broadcast(String.format(format:"User[%d] made an invalid guess", clientId), clientId);
105                     return true;
106                 }
107             }
108         }
109
110         if (message.equalsIgnoreCase(anotherString:"start")) {
111             gameActive = true;
112             secretNumber = new Random().nextInt(bound:10) + 1;
113             broadcast(message:"Number Guessing Game! Guess a number. Make sure to type 'guess:' and then your guess." , clientId);
114             return true;
115         }
116
117         if (message.equalsIgnoreCase(anotherString:"stop")) {
118             gameActive = false;
119             broadcast(message:"Number Guessing Game Exited.", clientId);
120             return true;
121         }
122         return false;
123     }
```

This code shows how I have edited the processCommand method in order to implement the game.

Checklist Items (1)

#2 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.

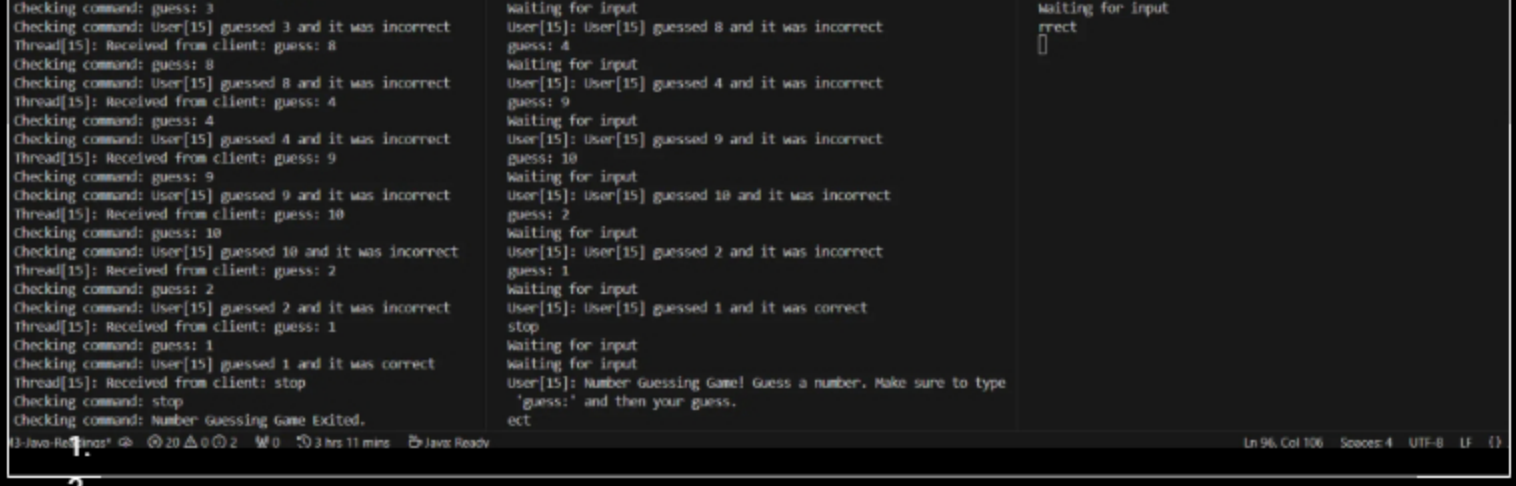
```
Starting Server
Server is listening on port 3000
waiting for next client
waiting for next client
Client connected
Thread[15]: Thread created
Thread[15]: Thread starting
waiting for next client
Client connected
Thread[16]: Thread created
Thread[16]: Thread starting
Thread[15]: Received from client: hi
Checking command: hi
Thread[16]: Received from client: hello
Checking command: hello
Thread[15]: Received from client: start
Checking command: start
Checking command: Number Guessing Game! Guess a number. Make
 sure to type 'guess:' and then your guess.
Thread[15]: Received from client: guess: 5
Checking command: guess: 5
Checking command: User[15] guessed 5 and it was incorrect
Thread[15]: Received from client: guess: 3
```

```
PS C:\Users\Shreya\Desktop\IT114\sb57-IT114-006\Module-4> java Pa
rt3HW\Client.java

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
hi
Waiting for input
User[15]: hi
User[16]: hello
start
Waiting for input
User[15]: Number Guessing Game! Guess a number. Make sure to type
 'guess:' and then your guess.
guess: 5
Waiting for input
User[15]: User[15] guessed 5 and it was incorrect
guess: 3
Waiting for input
User[15]: User[15] guessed 3 and it was incorrect
guess: 8
```

```
PS C:\Users\Shreya\Desktop\IT114\sb57-IT114-006\Module-4> jav
a Part-3\Client.java

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
User[15]: hi
hello
Waiting for input
User[16]: hello
User[15]: Number Guessing Game! Guess a number. Make sure to
type 'guess:' and then your guess.
User[15]: User[15] guessed 5 and it was incorrect
User[15]: User[15] guessed 3 and it was incorrect
User[15]: User[15] guessed 8 and it was incorrect
User[15]: User[15] guessed 4 and it was incorrect
User[15]: User[15] guessed 9 and it was incorrect
User[15]: User[15] guessed 10 and it was incorrect
ng Game! Guess a number. Make sure to type 'guess:' and then
your guess.
guess: 5
```



1.

2.

This screenshot shows me using the server and client to communicate with 2 clients connected. I show the full implementation of the game using my terminal.

1.

2.

Checklist Items (1)

#1 Output is clearly shown and captioned

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:

I chose the Coin Toss Command feature to implement:

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 added the commands flip coin and coin toss to activate the coin toss feature. My code screenshots show the output mentioning who did what and got what result.

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

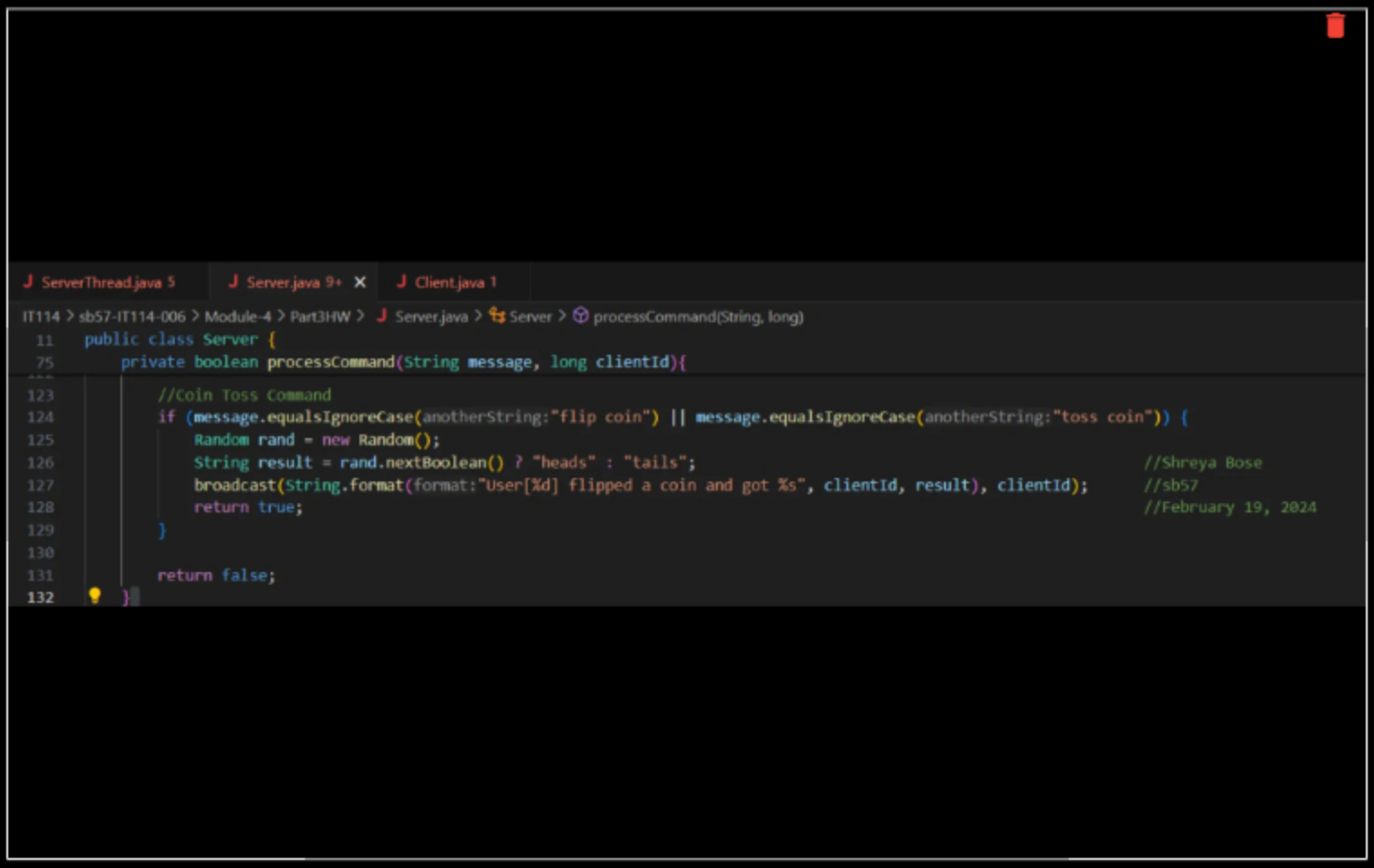
*The checkboxes are for your own tracking

#	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

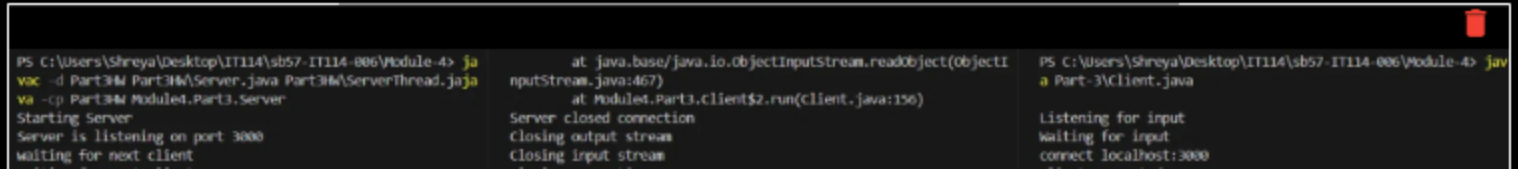
Small Medium Large

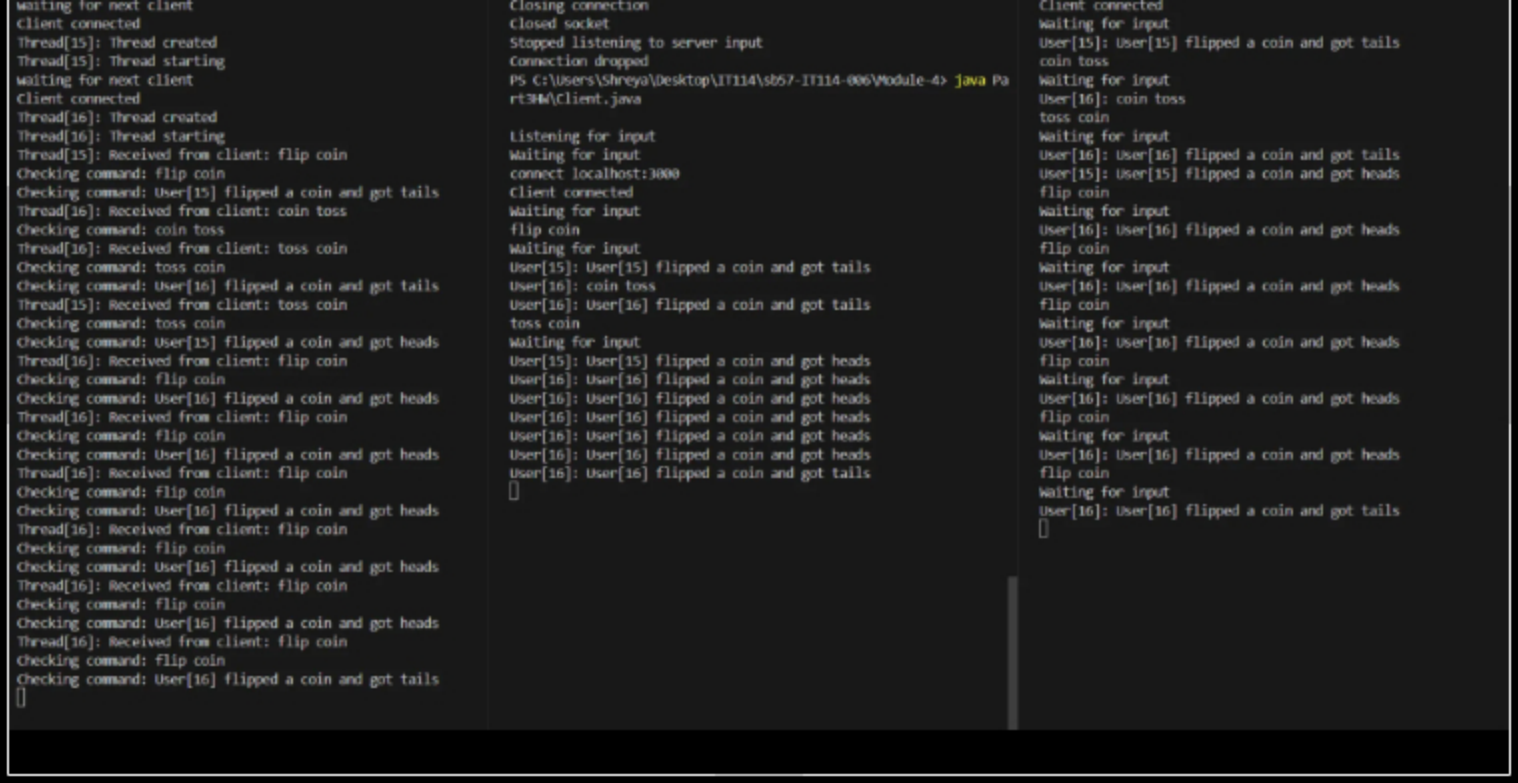


This screenshot shows the edits I made to the processCommand method in order to implement the coin toss command feature.

Checklist Items (1)

#2 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.





This screenshot shows me using the flip coin and toss coin commands to activate the coin toss command implementation through the terminal.

Checklist Items (1)

#1 Output is clearly shown and captioned

Misc (2 pts.)

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 difficulty compiling and running the Server.java and ServerThread.java files at first through the command line but I was able to debug it.

Task #2 - Points: 1

✕ COLLAPSE ✕

Text: Pull request link

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

URL #1
https://github.com/sb57-shreya/sb57-IT114-006/pull/7

End of Assignment