Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-006-S2024/it114-sockets-part-1-3-checkpoint/grade/bm47

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

Submissions:

Submission Selection

1 Submission [active] 2/21/2024 10:31:39 PM

Instructions

COLLAPSE

1.
- 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)
    1. 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
    2. 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

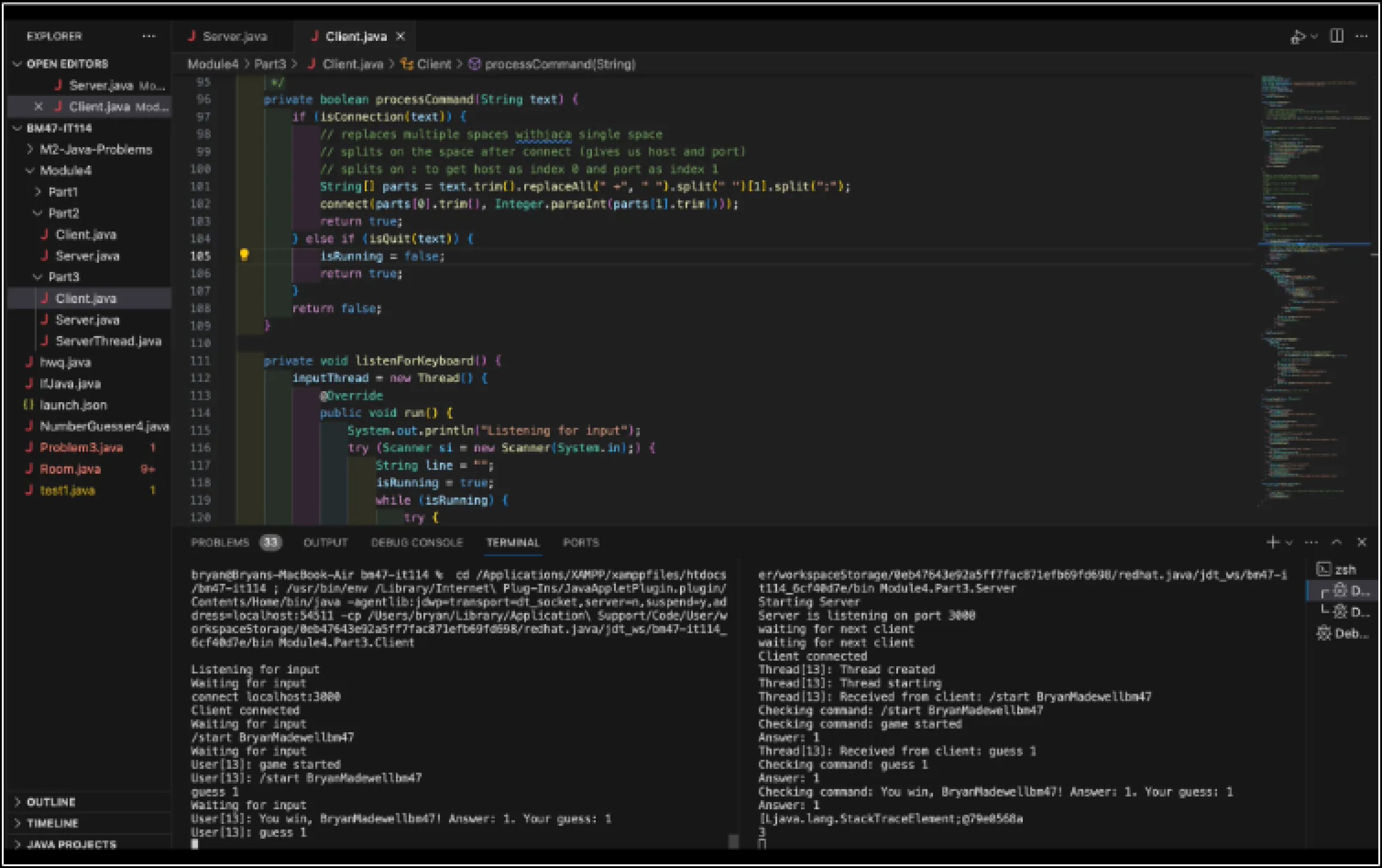
Checklist *The checkboxes are for your own tracking

#	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
■ #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



```java
     */
    private boolean processCommand(String text) {
        if (isConnection(text)) {
            // replaces multiple spaces withjaca single space
            // splits on the space after connect (gives us host and port)
            // splits on : to get host as index 0 and port as index 1
            String[] parts = text.trim().replaceAll(" +", " ").split(" ")[1].split(":");
            connect(parts[0].trim(), Integer.parseInt(parts[1].trim()));
            return true;
        } else if (isQuit(text)) {
            isRunning = false;
            return true;
        }
        return false;
    }

    private void listenForKeyboard() {
        inputThread = new Thread() {
            @Override
            public void run() {
                System.out.println("Listening for input");
                try (Scanner si = new Scanner(System.in);) {
                    String line = "";
                    isRunning = true;
                    while (isRunning) {
                        try {
```

```
bryan@Bryans-MacBook-Air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdocs
/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin/
Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,ad
dress=localhost:54511 -cp /Users/bryan/Library/Application\ Support/Code/User/w
orkspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-it114_
6cf40d7e/bin Module4.Part3.Client

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
/start BryanMadewellbm47
Waiting for input
User[13]: game started
User[13]: /start BryanMadewellbm47
guess 1
Waiting for input
User[13]: You win, BryanMadewellbm47! Answer: 1. Your guess: 1
User[13]: guess 1
```

```
er/workspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-i
t114_6cf40d7e/bin Module4.Part3.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]: Received from client: /start BryanMadewellbm47
Checking command: /start BryanMadewellbm47
Checking command: game started
Answer: 1
Thread[13]: Received from client: guess 1
Checking command: guess 1
Answer: 1
Checking command: You win, BryanMadewellbm47! Answer: 1. Your guess: 1
Answer: 1
[Ljava.lang.StackTraceElement;@79e0568a
3
```

In this caption, I have the server/terminal instance shown using the number guesser game that I have implemented from the choice of things to implement. In the terminal, it shows the game working, and I also implemented something that allows the user to choose their name upon startup to easily identify who is who. On the left, it is shown that I have all 3 parts downloaded onto my machine, along with each respective server and client file.

Checklist Items (4)

#1 Server terminal/instance is clearly shown/noted

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

#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 implemented the number guesser feature for my first option. The directions are listed below. Firstly, I created a boolean named "gamestarted" to see whether or not the game was started using the game command. Then, I listed all of the variables needed for the game such as randomNumber and a string for the name. If gameStarted == true, the game then selected a random number up to index 6, so numbers up to 5 can be the correct answers. I implemented the name into the /start command, using the split feature. The "/start Bryan" command would split the command into two separate terms, /start being index 0 and Bryan being index 1. The code selects index 1 as the name and then addresses everyone using this index so they know who is who.

- 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)

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

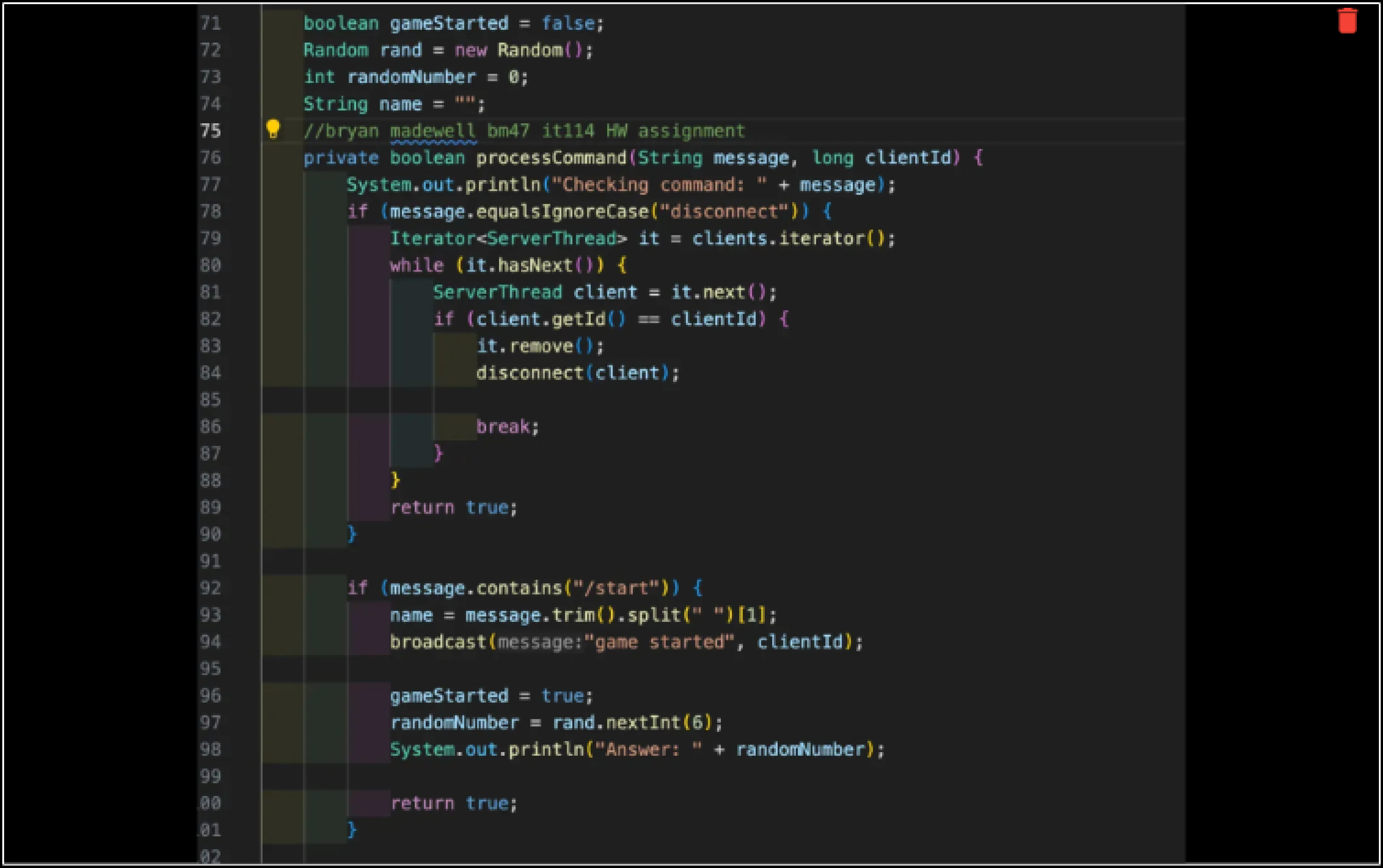
Checklist *The checkboxes are for your own tracking

#	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

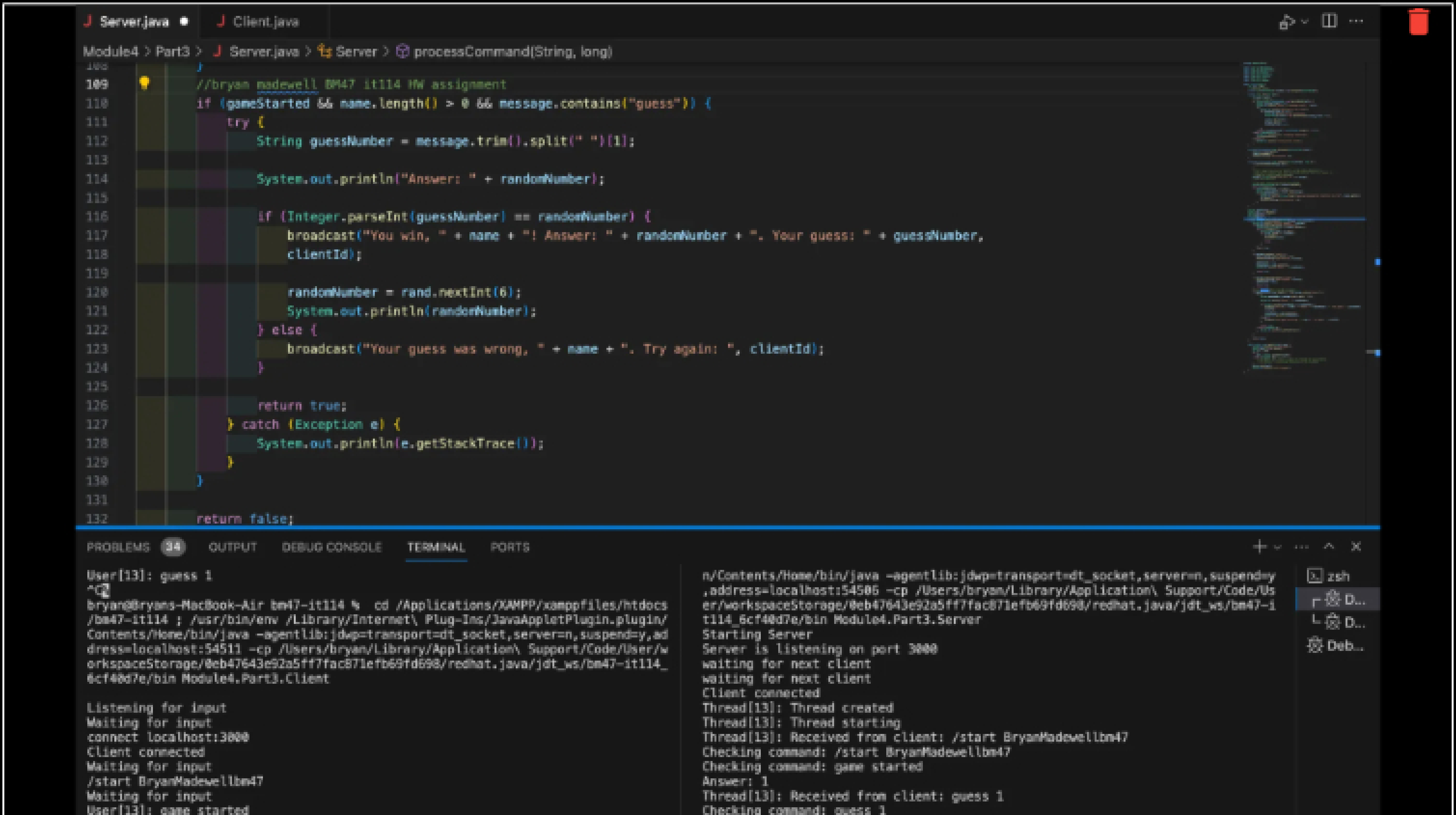


```java
    boolean gameStarted = false;
    Random rand = new Random();
    int randomNumber = 0;
    String name = "";
    //bryan madewell bm47 it114 HW assignment
    private boolean processCommand(String message, long clientId) {
        System.out.println("Checking command: " + message);
        if (message.equalsIgnoreCase("disconnect")) {
            Iterator<ServerThread> it = clients.iterator();
            while (it.hasNext()) {
                ServerThread client = it.next();
                if (client.getId() == clientId) {
                    it.remove();
                    disconnect(client);

                    break;
                }
            }
            return true;
        }

        if (message.contains("/start")) {
            name = message.trim().split(" ")[1];
            broadcast(message:"game started", clientId);

            gameStarted = true;
            randomNumber = rand.nextInt(6);
            System.out.println("Answer: " + randomNumber);

            return true;
        }

```

This screenshot shows some of the code used for the number guesser game feature that I added. It shows some of the methods used, and variables used. In another screenshot I will show the code output again as I did in the first screenshot on this assignment.

Checklist Items (1)

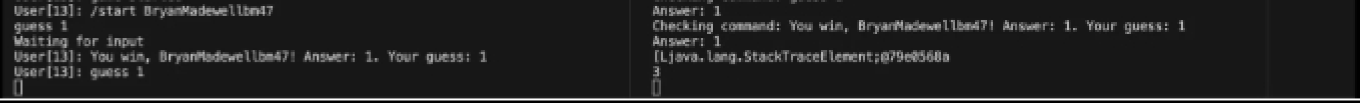
#2 Code shows relevant snippets that accomplish feature, UCID and date are present in all code screenshots. Relevant captions are included for each screenshot of the code.



```java
        }
        //bryan madewell BM47 it114 HW assignment
        if (gameStarted && name.length() > 0 && message.contains("guess")) {
            try {
                String guessNumber = message.trim().split(" ")[1];

                System.out.println("Answer: " + randomNumber);

                if (Integer.parseInt(guessNumber) == randomNumber) {
                    broadcast("You win, " + name + "! Answer: " + randomNumber + ". Your guess: " + guessNumber,
                    clientId);

                    randomNumber = rand.nextInt(6);
                    System.out.println(randomNumber);
                } else {
                    broadcast("Your guess was wrong, " + name + ". Try again: ", clientId);
                }

                return true;
            } catch (Exception e) {
                System.out.println(e.getStackTrace());
            }
        }

        return false;
```

```
User[13]: guess 1
^C
bryan@Bryans-MacBook-Air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdocs
/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin/
Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,ad
dress=localhost:54511 -cp /Users/bryan/Library/Application\ Support/Code/User/w
orkspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-it114_
6cf40d7e/bin Module4.Part3.Client

Listening for input
Waiting for input
connect localhost:3000
Client connected
Waiting for input
/start BryanMadewellbm47
Waiting for input
User[13]: game started
```

```
n/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y
,address=localhost:54506 -cp /Users/bryan/Library/Application\ Support/Code/Us
er/workspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-i
t114_6cf40d7e/bin Module4.Part3.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]: Received from client: /start BryanMadewellbm47
Checking command: /start BryanMadewellbm47
Checking command: game started
Answer: 1
Thread[13]: Received from client: guess 1
Checking command: guess 1
```



```
User[13]: /start BryanMadewellbm47
guess 1
Waiting for input
User[13]: You win, BryanMadewellbm47! Answer: 1. Your guess: 1
User[13]: guess 1
[]
```

```
Answer: 1
Checking command: You win, BryanMadewellbm47! Answer: 1. Your guess: 1
Answer: 1
[Ljava.lang.StackTraceElement;@79e0568a
3
[]
```

This screenshot shows the number guesser game working, using the /start command to start the game, and guessing by using the word "guess"

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 picked the coin flip game. I implemented it in a similar way as I did with the number guesser game. The directions are down below. I used the /flip command to flip the coin, as the /start command is being used by the number guesser game.

- 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)

● COLLAPSE

Task #2 - Points: 1

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

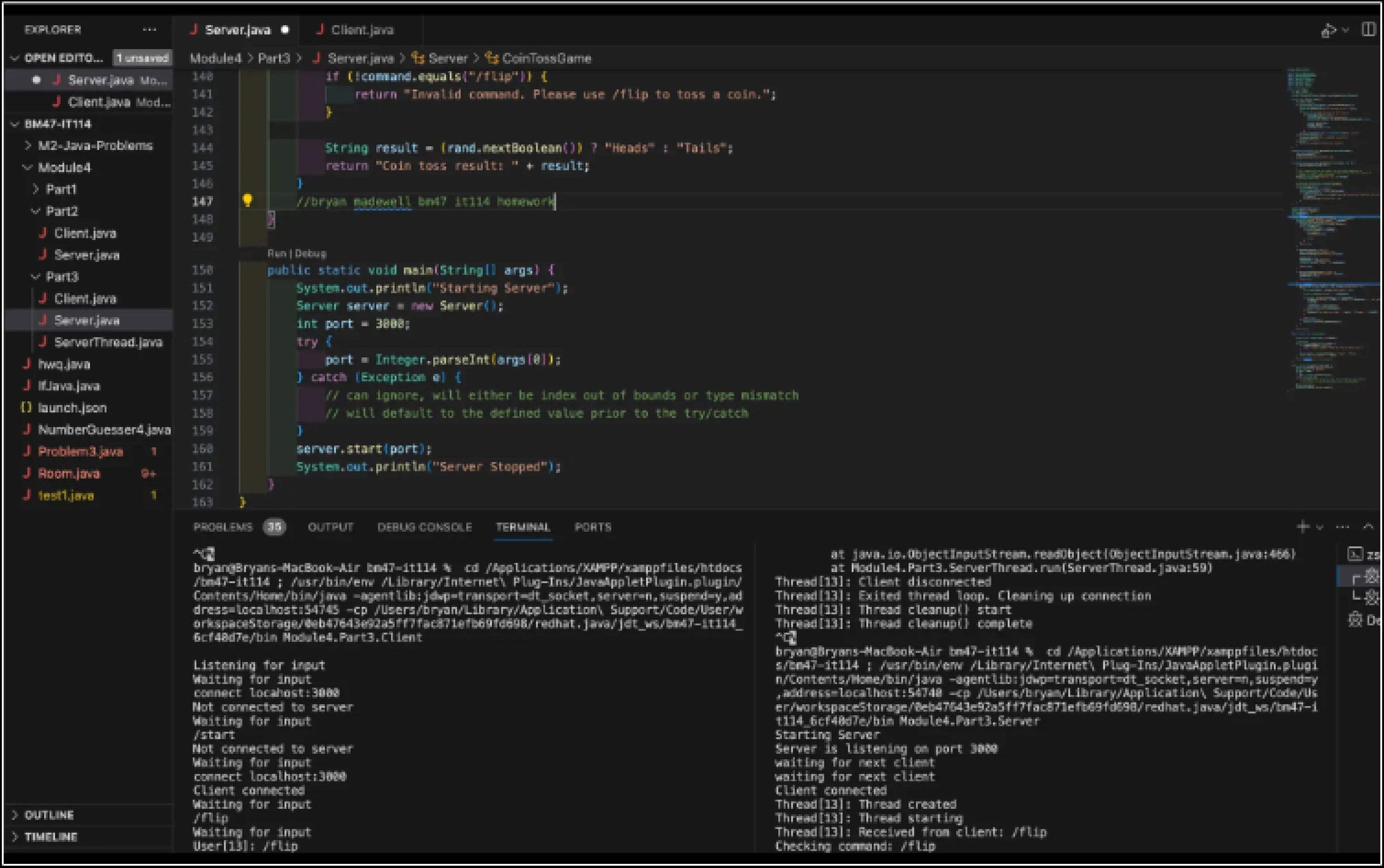
Details:
Add screenshots of the relevant code changes AND relevant output during runtime

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



```java
            if (!command.equals("/flip")) {
                return "Invalid command. Please use /flip to toss a coin.";
            }

            String result = (rand.nextBoolean()) ? "Heads" : "Tails";
            return "Coin toss result: " + result;
        }
        //bryan madewell bm47 it114 homework
    }

    public static void main(String[] args) {
        System.out.println("Starting Server");
        Server server = new Server();
        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");
    }
}
```

```
^C
bryan@Bryans-MacBook-Air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdocs
/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin/
Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,ad
dress=localhost:54745 -cp /Users/bryan/Library/Application\ Support/Code/User/w
orkspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-it114_
6cf40d7e/bin Module4.Part3.Client

Listening for input
Waiting for input
connect localhost:3000
Not connected to server
Waiting for input
/start
Not connected to server
Waiting for input
connect localhost:3000
Client connected
Waiting for input
/flip
Waiting for input
User[13]: /flip
```

```
        at java.io.ObjectInputStream.readObject(ObjectInputStream.java:466)
        at Module4.Part3.ServerThread.run(ServerThread.java:59)
Thread[13]: Client disconnected
Thread[13]: Exited thread loop. Cleaning up connection
Thread[13]: Thread cleanup() start
Thread[13]: Thread cleanup() complete
^C
bryan@Bryans-MacBook-Air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdoc
s/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugi
n/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y
,address=localhost:54740 -cp /Users/bryan/Library/Application\ Support/Code/Us
er/workspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_ws/bm47-i
t114_6cf40d7e/bin Module4.Part3.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]: Received from client: /flip
Checking command: /flip
```

I implemented the coinflip game using its own method. I had some issues getting the result to broadcast to all of the clients connected to the server, but it does work as intended for the most part. Most of the code is located at the top of the screenshot.

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:

This was my first time working with java sockets, so I definitely struggled. Prior to starting the assignment, I watched a few videos online about how they work as this is completely new to me. I got a general idea, and began working on the assignment. Some challenges I faced were getting the message to broadcast to all clients, and also just creating the games in general. I definitely need to work on sockets more, but I do find them quite interesting.

COLLAPSE

Task #2 - Points: 1

Text: Pull request link

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

URL #1
Missing URL

End of Assignment