Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-milestone-2-chatroom-2024/grade/ns87

IT114-002-S2024 - [IT114] Milestone 2 Chatroom 2024

Submissions:

Submission Selection

1 Submission [active] 4/1/2024 7:55:20 PM

Instructions

COLLAPSE

Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizQG4/view
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone2 branch
Create a pull request from Milestone2 to main and keep it open until you get the output PDF from this assignment.
Gather the evidence of feature completion based on the below tasks.
Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
Run the necessary git add, commit, and push steps to move it to GitHub
Complete the pull request that was opened earlier
Upload the same output PDF to Canvas

Branch name: Milestone2

Tasks: 12 Points: 10.00

Demonstrate Usage of Payloads (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of your Payload class and subclasses and PayloadType

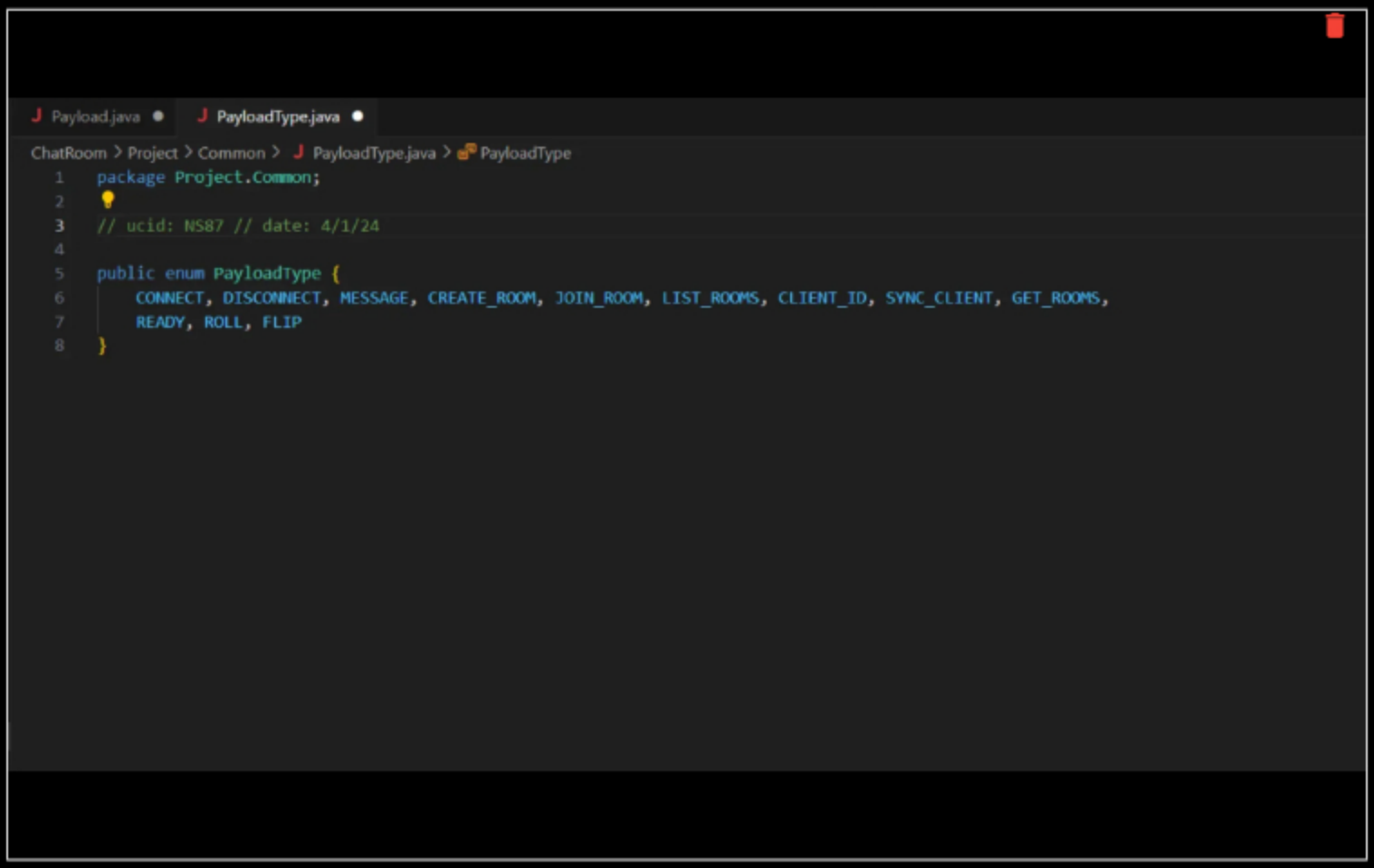
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Payload, equivalent of RollPayload, and any others
#2	1	Screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



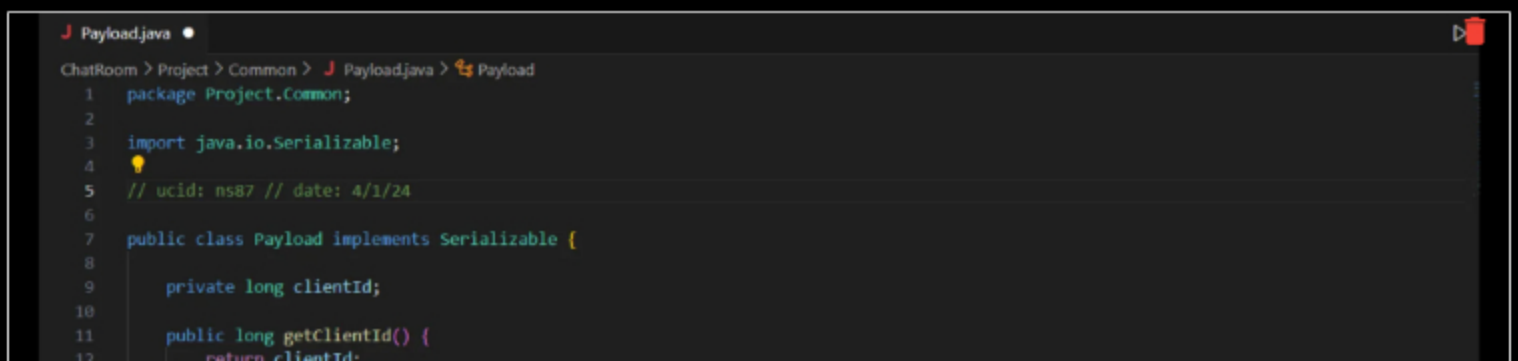
PayloadType

Checklist Items (3)

#1 Payload, equivalent of RollPayload, and any others

#2 Screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned



```java
    }

    public void setClientId(long clientId) {
        this.clientId = clientId;
    }

    // read https://www.baeldung.com/java-serial-version-uid
    private static final long serialVersionUID = 1L;// change this if the class changes

    /**
     * Determines how to process the data on the receiver's side
     */
    private PayloadType payloadType;

    public PayloadType getPayloadType() {
        return payloadType;
    }

    public void setPayloadType(PayloadType payloadType) {
        this.payloadType = payloadType;
    }
```

Payload

Checklist Items (3)

#1 Payload, equivalent of RollPayload, and any others

#2 Screenshots should include ucid and date comment

#3 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the payloads being debugged/output to the terminal

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Demonstrate flip
#2	1	Demonstrate roll (both versions)
#3	1	Demonstrate formatted message along with any others
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

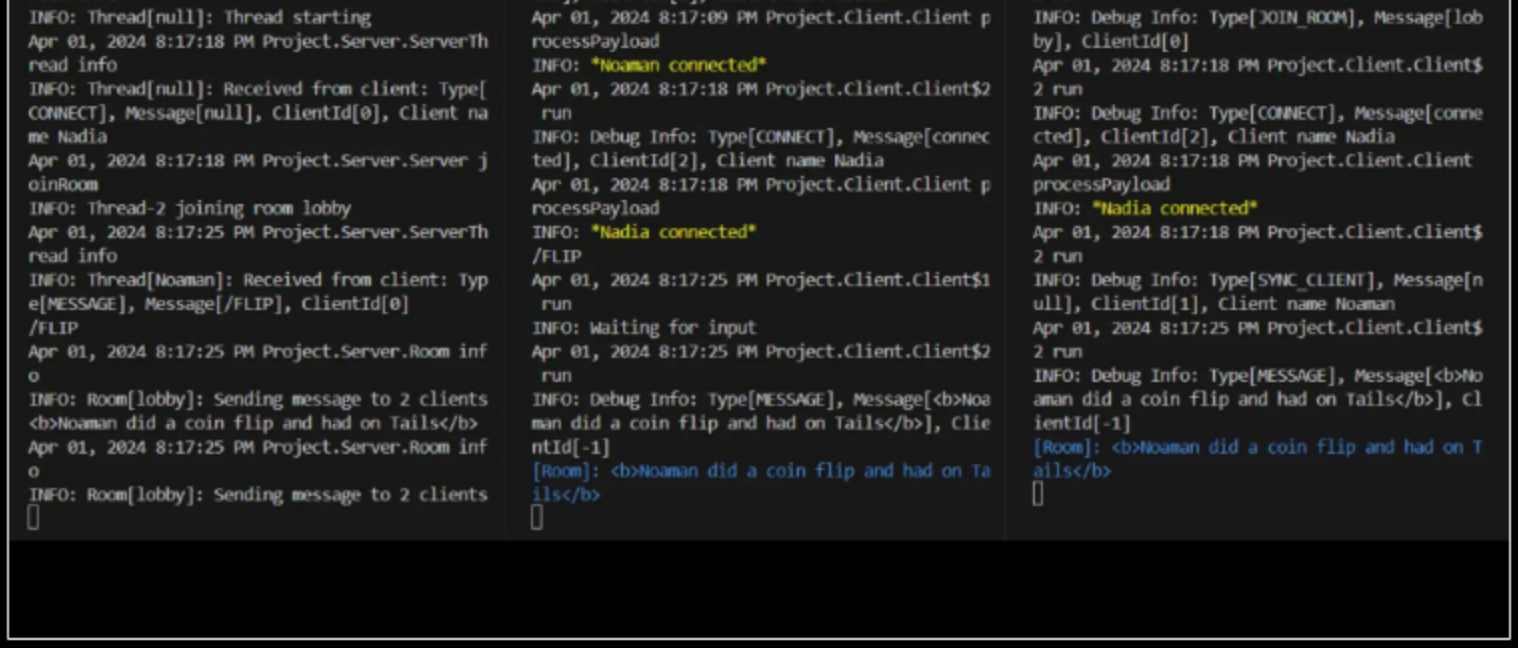
Gallery Style: Large View

Small Medium Large

```
Apr 01, 2024 8:17:18 PM Project.Server.Server start
INFO: Client connected
Apr 01, 2024 8:17:18 PM Project.Server.ServerThread info
INFO: Thread[null]: Thread created
Apr 01, 2024 8:17:18 PM Project.Server.ServerThread info
```

```
Apr 01, 2024 8:17:09 PM Project.Client.Client$2 run
INFO: Debug Info: Type[JOIN_ROOM], Message[lobby], ClientId[0]
Apr 01, 2024 8:17:09 PM Project.Client.Client$2 run
INFO: Debug Info: Type[CONNECT], Message[connected], ClientId[1], Client name Noaman
```

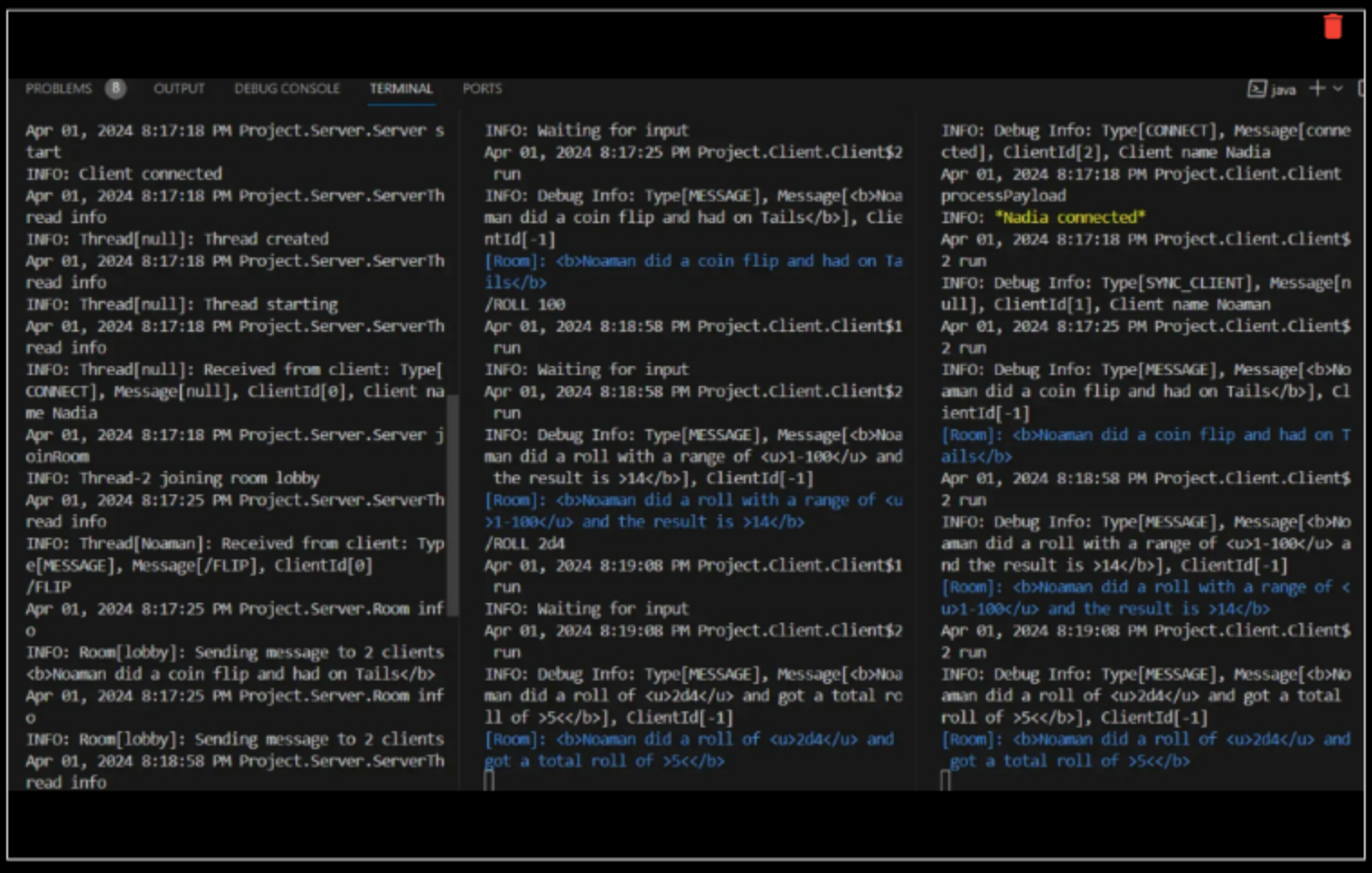
```
2 run
INFO: Debug Info: Type[CLIENT_ID], Message[null], ClientId[2], Client name Nadia
Apr 01, 2024 8:17:18 PM Project.Client.Client processPayload
INFO: My Client Id is 2
Apr 01, 2024 8:17:18 PM Project.Client.Client$2 run
```



Demonstrate flip

Checklist Items (1)

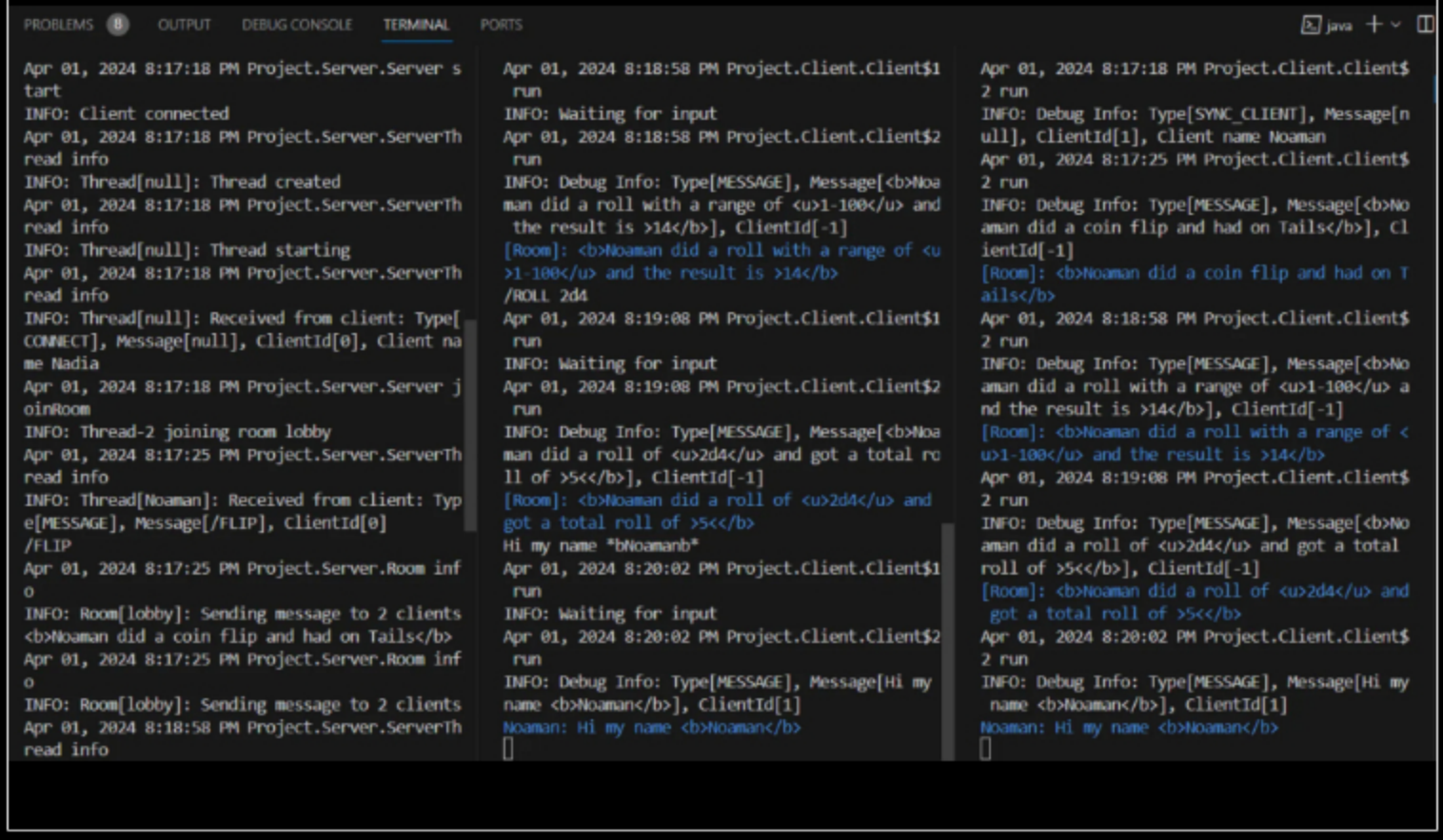
#1 Demonstrate flip



Demonstrate roll (both versions)

Checklist Items (1)

#2 Demonstrate roll (both versions)



Demonstrate formatted message along with any others

Checklist Items (2)

#3 Demonstrate formatted message along with any others

#4 Each screenshot should be clearly captioned

COLLAPSE

Task #3 - Points: 1

Text: Explain the purpose of payloads and how your flip/roll payloads were made

Response:

A payload typically refers to the data transmitted over a network or between different system components. It can also denote the data processed or manipulated by a program.

Flip Payload:

The flip payload activates upon receiving the "FLIP" command followed by user input.
When a client sends a message starting with "/FLIP", the server interprets it as a flip command.
Within the processCommands method, the server generates a random number (0 or 1) to simulate a coin flip.
Based on the random result (heads or tails), the server sends a formatted message to all room clients, indicating the flip's outcome.

Roll Payload:

The roll payload is triggered by the "ROLL" command, followed by user-defined parameters for dice rolling.

When a client sends a message starting with "/ROLL", the server processes it as a roll command.
The server parses the user input to determine the roll type (e.g., number of dice, sides of dice) or a single roll within a specified range.
Subsequently, it calculates the roll result(s) based on the provided parameters and sends a formatted message to all room clients, displaying the roll outcome.

Demonstrate Roll Command (2 pts.)

^COLLAPSE ^

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshot of the following items

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#
#2	1	ServerThread code receiving the payload and passing it to the Room
#3	1	Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients
#4	1	Code screenshots should include ucid and date comment
#5	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
//UCID:NS87 DATE:04.1.24

    case "ROLL":
    int total = 0;
    try {
    String[] rollParts = message.trim().split(regex:" ");
    if (rollParts.length >= 2 && rollParts[1].contains(s:"d")) {
    String[] rollParams = rollParts[1].split(regex:"d");
    int numberOfDice = Integer.parseInt(rollParams[0]);
    int sidesOfDice = Integer.parseInt(rollParams[1]);
    for (int i = 0; i < numberOfDice; i++) {
    int rollDice = (int) (Math.random() * sidesOfDice) + 1;
    total += rollDice;
}
    sendMessage(sender:null, String.format(format:"<b>%s did a roll of <u>%sd%s</u> and got a total roll of >%s<</b>", client.getClientName(), numberOfDice, sidesOfDice, total));
    } else if (rollParts.length >= 2) {
    int value = Integer.parseInt(rollParts[1]);
    int singleDiceRoll = (int) (Math.random() * value) + 1;
    sendMessage(sender:null, String.format(format:"<b>%s did a roll with a range of <u>1-%s</u> and the result is >%s</b>", client.getClientName(), value, singleDiceRoll));
    } else {
    client.sendMessage(-1, message:"<b>invalid input</b>");
    }
    } catch (NumberFormatException e) {
    client.sendMessage(-1, message:"<b>invalid input</b>");
    }
    break;



    default:
        wasCommand = false;
        break;
}
```

Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients

Checklist Items (4)

#1 Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#

#3 Room handling the roll action correctly for both scenarios (/roll # and /roll #d#) including the message going back out to all clients

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
    private Room currentRoom;
    private Logger logger = Logger.getLogger(ServerThread.class.getName());

    private void info(String message) {
        logger.info(String.format(format:"Thread[%s]: %s", getClientName(), message));
    }
// ucid: NS87  date: 4/1/24
    public ServerThread(Socket myClient/* , Room room */) {
        info(message:"Thread created");
        // get communication channels to single client
        this.client = myClient;
        // this.currentRoom = room;

    }

    protected void setClientId(long id) {
        clientId = id;
        if (id == Constants.DEFAULT_CLIENT_ID) {
            logger.info(TextFX.colorize(text:"Client id reset", Color.WHITE));
```

ServerThread code receiving the payload and passing it to the Room

Checklist Items (1)

#2 ServerThread code receiving the payload and passing it to the Room

```
23
24      private void sendConnect() throws IOException {
25          ConnectionPayload p = new ConnectionPayload(isConnected:true);
26
27          p.setClientName(clientName);
28          out.writeObject(p);
```

```java
    }

    private void sendMessage(String message) throws IOException {
        Payload p = new Payload();                    // NS87 4/1/24
        if(processClientCommand(message)){
            return;
        }
        p.setPayloadType(PayloadType.MESSAGE);
        p.setMessage(message);
        // no need to send an identifier, because the server knows who we are
        // p.setClientName(clientName);
        out.writeObject(p);
    }

    // end send methods
    private void listenForKeyboard() {
        inputThread = new Thread() {
            @Override
            public void run() {
```

Client code

Checklist Items (1)

#1 Client code that captures the command and converts it to a RollPayload (or equivalent) for both scenarios /roll # and /roll #d#

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the two different roll formats are handled and how the message flows from the client, to the Room, and shared with all other users

Response:

Rolling a Specified Number of Dice with a Specified Number of Sides:

The format for this type of roll is "ROLL XdY", where X is the number of dice and Y is the number of sides on each die.
When a client sends a message in this format, such as "/ROLL 2d6" (roll 2 six-sided dice), the server parses the message to extract the number of dice and sides.
It then simulates rolling each die, calculates the total, and sends a formatted message to all clients in the room, displaying the total roll outcome.
Rolling Within a Specified Range:

The format for this type of roll is "ROLL X", where X is the upper limit of the range (from 1 to X).
When a client sends a message in this format, such as "/ROLL 20" (roll within the range of 1-20), the server parses the message to extract the upper limit of the range.
It then generates a random number within the specified range, simulating a roll, and sends a formatted message to all clients in the room, displaying the roll outcome.
Message Flow:

Client to Room:

A client sends a message containing the roll command ("/ROLL") and the desired parameters (either XdY or X for the range).

The message is received by the Room class, specifically the sendMessage method, where it is processed.
Room Processing:

Inside the sendMessage method, the server checks if the message is a roll command using the processCommands method.
If it's a roll command, the server parses the message to determine the roll type (number of dice or range) and performs the corresponding roll calculation.
Sharing with Other Users:

After calculating the roll outcome, the server formats a message containing the result.
This formatted message is then sent to all clients in the room using the sendMessage method, ensuring that all users receive and see the roll outcome.

Demonstrate Flip Command (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of the following items

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Client code that captures the command and converts it to a payload
#2	1	ServerThread receiving the payload and passing it to the Room
#3	1	Room handling the flip action correctly
#4	1	Code screenshots should include ucid and date comment
#5	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
    private Room currentRoom;
    private Logger logger = Logger.getLogger(ServerThread.class.getName());

    private void info(String message) {
        logger.info(String.format(format:"Thread[%s]: %s", getClientName(), message));
    }
// ucid: NS87  date: 4/1/24
    public ServerThread(Socket myClient/* , Room room */) {
        info(message:"Thread created");
        // get communication channels to single client
        this.client = myClient;
        // this.currentRoom = room;
```

```
    }

    protected void setClientId(long id) {
        clientId = id;
        if (id == Constants.DEFAULT_CLIENT_ID) {
            logger.info(TextFX.colorize(text:"Client id reset", Color.WHITE));
```

ServerThread receiving the payload and passing it to the Room

Checklist Items (4)

#1 Client code that captures the command and converts it to a payload

#2 ServerThread receiving the payload and passing it to the Room

#4 Code screenshots should include ucid and date comment

#5 Each screenshot should be clearly captioned

```
         */

        //UCID:NS87 DATE:04.1.24
            case "FLIP":
            int coin = (int) (Math.random() * 2);
            if (coin == 0) {
            sendMessage(sender:null, String.format(format:"<b>%s did a coin flip and had Heads</b>", client.getClientName()));
             } else if (coin == 1) {
            sendMessage(sender:null, String.format(format:"<b>%s did a coin flip and had on Tails</b>", client.getClientName()));
             }
            break;
```

Room handling the flip action correctly

Checklist Items (1)

#3 Room handling the flip action correctly

```java
23
24      private void sendConnect() throws IOException {
25          ConnectionPayload p = new ConnectionPayload(isConnected:true);
26
27          p.setClientName(clientName);
28          out.writeObject(p);
29      }
30
31      private void sendMessage(String message) throws IOException {
32          Payload p = new Payload();                      // NS87 4/1/24
33          if(processClientCommand(message)){
34              return;
35          }
36          p.setPayloadType(PayloadType.MESSAGE);
37          p.setMessage(message);
38          // no need to send an identifier, because the server knows who we are
39          // p.setClientName(clientName);
40          out.writeObject(p);
41      }
42
43      // end send methods
44      private void listenForKeyboard() {
45          inputThread = new Thread() {
46              @Override
47              public void run() {
```

Client code

Checklist Items (1)

#1 Client code that captures the command and converts it to a payload

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic in how the flip command is handled and processed and how the message flows from the client, to the Room, and shared with all other users

Response:

Client Initiates Flip Command:

A client sends a message containing the flip command ("/FLIP") to the chat room.
Room Receives and Processes the Flip Command:

The Room class, specifically the sendMessage method, receives the message containing the flip command.
Inside the sendMessage method, the server checks if the message is a flip command using the processCommands method.
Since it is a flip command, the server generates a random number (0 or 1) to simulate the result of a coin flip (heads or tails).
Outcome Shared with All Users:

After simulating the coin flip, the server formats a message indicating the outcome of the flip (heads or tails).
This formatted message is then sent to all clients in the room using the sendMessage method.

Demonstrate Formatted Messages (4 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of Room how the following formatting is processed from a message

Details:

Note: this processing is server-side

Slash commands are not valid solutions for this and will receive 0 credit

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Room code processing for bold
#2	1	Room code processing for italic
#3	1	Room code processing for underline
#4	1	Room code processing for color (at least R, G, B or support for hex codes)
#5	1	Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal
#6	1	Must not rely on the user typing html characters, but the output can be html characters
#7	1	Code screenshots should include ucid and date comment
#8	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
//UCID:NS87 DATE:04.1.24
//Formatted message output
String startTag = "";
String endTag = "";
int startIndex = -1;
int endIndex = -1;
boolean processed = true;
//UCID:NS87 DATE:04.1.24
while(processed){
    processed = false;
    startIndex = message.indexOf(str:"*b");
    endIndex = message.indexOf(str:"b*");
    //bold
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<b>";
        endTag = "</b>";
        message = message.substring(beginIndex:0, startIndex) + startTag
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
    //italic
    startIndex = message.indexOf(str:"*i");
    endIndex = message.indexOf(str:"i*");
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<i>";
        endTag = "</i>";
        message = message.substring(beginIndex:0, startIndex) + startTag
```

```
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
    //underline
    startIndex = message.indexOf(str:"*u");
    endIndex = message.indexOf(str:"u*");
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<u>";
        endTag = "</u>";
        message = message.substring(beginIndex:0, startIndex) + startTag
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
```

Room code processing for bold Room code processing for italic Room code processing for underline

Checklist Items (1)

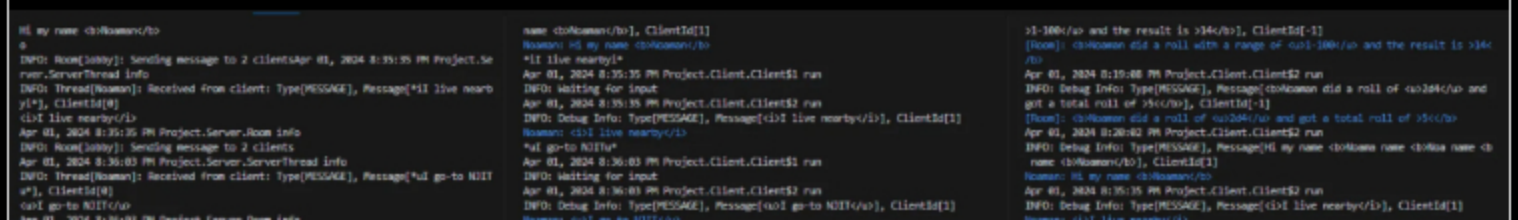
#1 Room code processing for bold

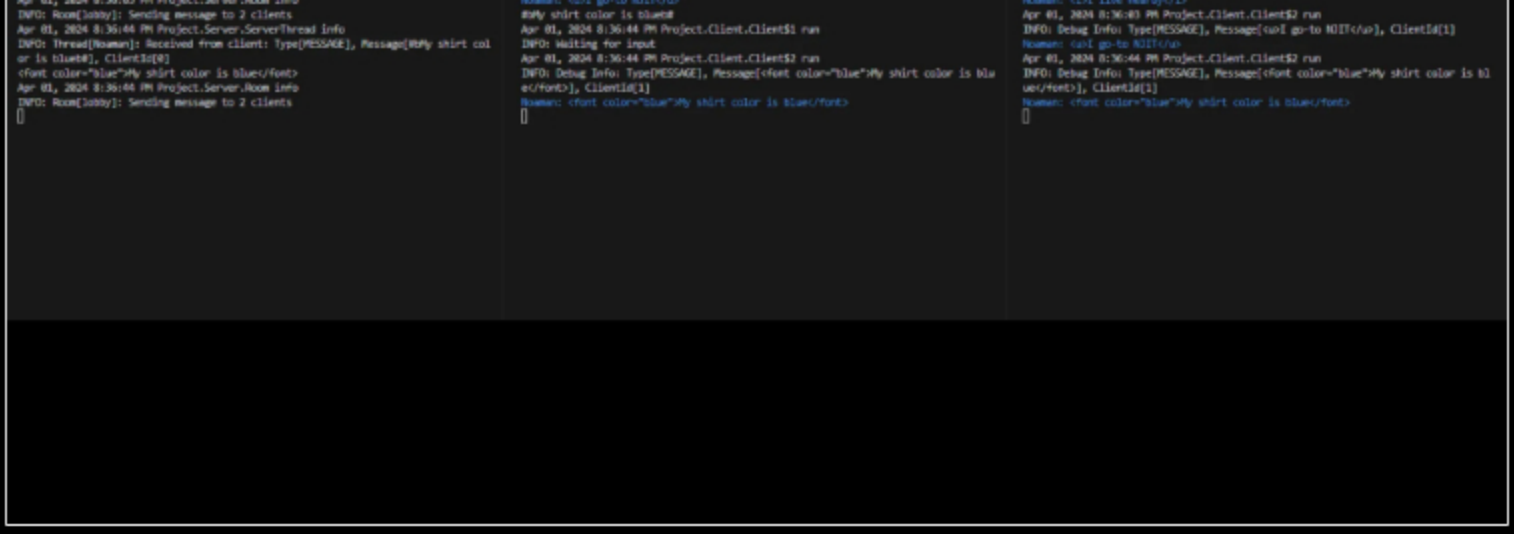
```
    //red
    startIndex = message.indexOf(str:"#r");
    endIndex = message.indexOf(str:"r#");
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<font color=\"red\">";
        endTag = "</font>";
        message = message.substring(beginIndex:0, startIndex) + startTag
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
    //green
    startIndex = message.indexOf(str:"#g");
    endIndex = message.indexOf(str:"g#");
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<font color=\"green\">";
        endTag = "</font>";
        message = message.substring(beginIndex:0, startIndex) + startTag
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
    //blue
    startIndex = message.indexOf(str:"#b");
    endIndex = message.indexOf(str:"b#");
    if(startIndex > -1 && endIndex > -1 && endIndex > startIndex+2){
        processed = true;
        startTag = "<font color=\"blue\">";
        endTag = "</font>";
        message = message.substring(beginIndex:0, startIndex) + startTag
        + message.substring(startIndex+2, endIndex) + endTag
        + message.substring(endIndex+2);
    }
```

Room code processing for color (at least R, G, B or support for hex codes)

Checklist Items (1)

#4 Room code processing for color (at least R, G, B or support for hex codes)





Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal

Checklist Items (2)

#5 Show each one working individually and one showing a combination of all of the formats and 1 color from the terminal

#6 Must not rely on the user typing html characters, but the output can be html characters

COLLAPSE

Task #2 - Points: 1

Text: Explain the following

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Which special characters translate to the desired effect
#2	1	How the logic works that converts the message to its final format

Response:

Bold Text (*b and b*):

The special characters *b and b* are used to denote the start and end of bold text, respectively.
For example, if a message contains *bHello, World!b*, the server interprets this as bold text and formats it accordingly in the final message.
The logic in the code checks for the presence of these special characters and wraps the text between them with HTML and tags, which represent bold formatting in HTML.
Italic Text (*i and i*):

Similarly, the special characters *i and i* mark the start and end of italicized text.
When the code encounters *iHello, World!i* in a message, it applies italic formatting by enclosing the text between these characters with HTML and tags.
Underline Text (*u and u*):

The special characters *u and u* are used for underlined text.
For instance, *uHello, World!u* is converted to HTML <u>Hello, World!</u> for underlining the text.
Colored Text (#r, #g, #b, and corresponding ending characters):

These special characters are used to specify text color in red (#r and r#), green (#g and g#), and blue (#b and b#). When a message contains #rHello, World!r#, it signifies red-colored text, and the code converts it to Hello, World! in HTML.
Similarly, #gHello, World!g# becomes Hello, World!, and #bHello, World!b# becomes Hello, World!.

Misc (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Add the pull request link for the branch

Details:
Note: the link should end with /pull/#

URL #1
https://github.com/Noaman4/-Milestone-Milestone-1/pull/2

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

One of the main challenges was managing the complexity of command handling within the chat server. Implementing commands like flipping a coin, rolling dice, and formatting text required careful logic to ensure proper functionality

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

Details:
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Wakatime

End of Assignment