

# Submission Worksheet

**CLICK TO GRADE**

<https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-milestone-2-chatroom-2024/grade/jeo29>

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

## Submissions:

Submission Selection

1 Submission [active] 4/3/2024 1:11:46 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 ^](#)

 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



```

121 //UCID:jeo29
122 //March 30,2024
123 private boolean processCommands(String message, ServerThread client) {
124     if (message.startsWith(prefix:"/roll")) {
125         String command = message.substring(beginIndex:6).trim();
126         try {
127             if (command.matches(regex:"\\d+d\\d+")) {
128                 // Format 2: /roll #d#
129                 String[] diceParts = command.split(regex:"d");
130                 int numberofDice = Integer.parseInt(diceParts[0].trim());
131                 int numberofSides = Integer.parseInt(diceParts[1].trim());
132                 StringBuilder result = new StringBuilder("Rolling " + numberofDice + "d" + numberofSides + ": ");
133                 for (int i = 0; i < numberofDice; i++) {
134                     int roll = (int) (Math.random() * numberofSides) + 1;
135                     result.append(roll);
136                     if (i < numberofDice - 1) {
137                         result.append(str:", ");
138                     }
139                 }
140             }
141         }
142     }
143 }
```

Payload, equivalent of RollPayload, and any others

Checklist Items (1)

#1 Payload, equivalent of RollPayload, and any others



```

121 //UCID:jeo29
122 //March 30,2024
123 private boolean processCommands(String message, ServerThread client) {
124     if (message.startsWith(prefix:"/roll")) {
125         String command = message.substring(beginIndex:6).trim();
126         try {
127             if (command.matches(regex:"\\d+d\\d+")) {
128                 // Format 2: /roll #d#
129                 String[] diceParts = command.split(regex:"d");
130                 int numberofDice = Integer.parseInt(diceParts[0].trim());
131                 int numberofSides = Integer.parseInt(diceParts[1].trim());
132                 StringBuilder result = new StringBuilder("Rolling " + numberofDice + "d" + numberofSides + ": ");
133                 for (int i = 0; i < numberofDice; i++) {
134                     int roll = (int) (Math.random() * numberofSides) + 1;
135                     result.append(roll);
136                     if (i < numberofDice - 1) {
137                         result.append(str:", ");
138                     }
139                 }
140             }
141         }
142     }
143 }
```

```

139     } <- #133-139 for (int i = 0; i < numberofdice; i++)
140         sendMessage(client, result.toString());
141     } else {
142         // Format 1: /roll 0 - X or 1 - X
143         String[] rangeParts = command.split(regex:"-");
144         int min = Integer.parseInt(rangeParts[0].trim());
145         int max = Integer.parseInt(rangeParts[1].trim());
146         int roll = (int) (Math.random() * (max - min + 1)) + min;
147         sendMessage(client, "Rolling " + min + " - " + max + ": " + roll);
148     } <- #141-148 else
149     return true;
150 } catch (Exception e) {
151     return false;
152 }
153 } else if (message.equals(anObject:"/flip")) {
154     return handleFlipCommand(client);
155 } else if (message.startsWith(prefix:"/format")) {
156     String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
157     sendMessage(client, formattedMessage);
158     return true;
159 } <- #155-159 else if (message.startsWith("/format"))
160     return false;
161 } <- #123-161 private boolean processCommands(String message, ServerThread ...

```

### Payload, equivalent of RollPayload, and any others

#### Checklist Items (1)

##### #1 Payload, equivalent of RollPayload, and any others

```

--+
153     } else if (message.equals(anObject:"/flip")) {
154         return handleFlipCommand(client);
155     } else if (message.startsWith(prefix:"/format")) {
156         String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
157         sendMessage(client, formattedMessage);
158         return true;
159     } <- #155-159 else if (message.startsWith("/format"))
160         return false;
161 } <- #123-161 private boolean processCommands(String message, ServerThread ...
162
163 //UCID:jeo29
164 //March 30,2024
165 private boolean handleFlipCommand(ServerThread client) {
166     int flip = (int) (Math.random() * 2);
167     String result = (flip == 0) ? "Heads" : "Tails";
168     sendMessage(client, "Coin flip: " + result);
169     return true;
170 } <- #165-170 private boolean handleFlipCommand(ServerThread client)

```

### Payload, equivalent of RollPayload, and any others

#### Checklist Items (1)

##### #1 Payload, equivalent of RollPayload, and any others

● Task #2 - Points: 1

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

▲ COLLAPSE ▲

## 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

```
152     }
153 } else if (message.equals(anObject:"/flip")) {
154     return handleFlipCommand(client);
155 } else if (message.startsWith(prefix:"/format")) {
156     String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
157     sendMessage(client, formattedMessage);
158     return true;
159 } <- #155-159 else if (message.startsWith("/format"))
160     return false;
161 } <- #123-161 private boolean processCommands(String message, ServerThread ...
162
163 //UCID:jeo29
164 //March 30,2024
165 private boolean handleFlipCommand(ServerThread client) {
166     int flip = (int) (Math.random() * 2);
167     String result = (flip == 0) ? "Heads" : "Tails";
168     sendMessage(client, "Coin flip: " + result);
169     return true;
170 } <- #165-170 private boolean handleFlipCommand(ServerThread client)
```

## Demonstrate flip

### Checklist Items (1)

#### #1 Demonstrate flip

```
121 //UCID:jeo29
122 //March 30,2024
123 private boolean processCommands(String message, ServerThread client) {
124     if (message.startsWith(prefix:"/roll")) {
125         String command = message.substring(beginIndex:6).trim();
126         try {
127             if (command.matches(regex:"\\d+d\\d+")) {
128                 // Format: 3+6=18
```

```

128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
    // Format 1: /roll X or 1 - X
    String[] diceParts = command.split(regex:"d");
    int numberOfDayDice = Integer.parseInt(diceParts[0].trim());
    int numberoSides = Integer.parseInt(diceParts[1].trim());
    StringBuilder result = new StringBuilder("Rolling " + numberOfDayDice + "d" + numberoSides + ": ");
    for (int i = 0; i < numberOfDayDice; i++) {
        int roll = (int) (Math.random() * numberoSides) + 1;
        result.append(roll);
        if (i < numberOfDayDice - 1) {
            result.append(str:", ");
        }
    } <- #133-139 for (int i = 0; i < numberOfDayDice; i++)
    sendMessage(client, result.toString());
} else {
    // Format 1: /roll 0 - X or 1 - X
    String[] rangeParts = command.split(regex:"-");
    int min = Integer.parseInt(rangeParts[0].trim());
    int max = Integer.parseInt(rangeParts[1].trim());
    int roll = (int) (Math.random() * (max - min + 1)) + min;
    sendMessage(client, "Rolling " + min + " - " + max + ": " + roll);
} <- #141-148 else
return true;
} catch (Exception e) {
    return false;
}
} else if (message.equals(anObject:"/flip")) {
    return handleFlipCommand(client);
} else if (message.startsWith(prefix:"/format")) {
    String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
    sendMessage(client, formattedMessage);
    return true;
} <- #155-159 else if (message.startsWith("/format"))
return false;
} <- #123-161 private boolean processCommands(String message, ServerThread ...

```

## Demonstrate roll (both versions)

### Checklist Items (1)

#### #2 Demonstrate roll (both versions)

```

IT114-450-Milestone.2 — java Project.server.Server — 98x30
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:41 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:41 AM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Apr 04, 2024 1:03:47 AM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[Jessica], Message[/flip]
Apr 04, 2024 1:03:47 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:47 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:47 AM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Tails]
Apr 04, 2024 1:03:50 AM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[Jessica], Message[/flip]
Apr 04, 2024 1:03:50 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:50 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:54 AM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Apr 04, 2024 1:03:54 AM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[Jessica], Message[/flip]
Apr 04, 2024 1:03:54 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:54 AM Project.server.Room sendMessage
INFO: Sending message to 1 clients
Apr 04, 2024 1:03:54 AM Project.server.ServerThread send
INFO: Sent payload: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Apr 04, 2024 1:03:56 AM Project.server.ServerThread run

IT114-450-Milestone.2 — java Project.client.Client — 103x31
Q Find Done
Apr 04, 2024 1:03:41 AM Project.client.Client$2 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Jessica: Coin flip: Heads
/flip
Apr 04, 2024 1:03:47 AM Project.client.Client$1 run
INFO: Waiting for input
Apr 04, 2024 1:03:47 AM Project.client.Client$2 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Tails]
Jessica: Coin flip: Tails
/flip
Apr 04, 2024 1:03:50 AM Project.client.Client$1 run
INFO: Waiting for input
Apr 04, 2024 1:03:50 AM Project.client.Client$2 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Jessica: Coin flip: Heads
/flip
Apr 04, 2024 1:03:54 AM Project.client.Client$1 run
INFO: Waiting for input
Apr 04, 2024 1:03:54 AM Project.client.Client$2 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Coin flip: Heads]
Jessica: Coin flip: Heads
/roll
Apr 04, 2024 1:03:56 AM Project.client.Client$1 run
INFO: Waiting for input
java.io.EOFException
at java.base/java.io.ObjectInputStream$BlockDataInputStream.peekByte(ObjectInputStream.java:323)
2)
at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1713)
at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:540)
at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:498)
at Project.client.Client$2.run(Client.java:561)

```

Heads/Tails

### Checklist Items (1)

#### #2 Demonstrate roll (both versions)

## #2 Demonstrate Roll (both versions)

The screenshot shows two terminal windows side-by-side. The left window is titled 'IT114-450-Milestone.2 — java Project.server.Server — 98x30' and the right window is titled 'IT114-450-Milestone.2 — java Project.client.Client — 103x31'. Both windows display log output from Java code. The server log shows a client connecting, sending a message, and then disconnecting. The client log shows it connecting to the server, receiving a message, and then disconnecting. The logs include timestamps and various system messages like 'INFO' and 'DEBUG'.

### Roll version

#### Checklist Items (1)

##### #2 Demonstrate roll (both versions)

#### Task #3 - Points: 1

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

#### Response:

The purpose of Payloads and how flip roll payloads were made is that in the 'Room' class payloads are implemented as text commands that clients send to perform the commands of flip or role. the flip command provides us with heads or tails. While the roll command generates a random number.

#### Demonstrate Roll Command (2 pts.)

**^ COLLAPSE ^**

#### Task #1 - Points: 1

**Text:** Screenshot of the following items

| #  | 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

```

107     |     System.out.println("Name set to " + clientName);
108   } |     return true;
109 } <- #103-110 if (text.startsWith("/name"))
110   |     return false;
111 } <- #102-112 private boolean isName(String text)
112
113
114 /**
115 * Controller for handling various text commands from the client
116 * <p>
117 * Add more here as needed
118 * </p>
119 *
120 * @param text
121 * @return true if a text was a command or triggered a command
122 */
123 @Deprecated // removing in Milestone3
124 private boolean processClientCommand(String text) throws IOException {
125     if (isConnection(text)) {
126         if (clientName.isBlank()) {
127             System.out.println("You must set your name before you can connect via: /name your_name");
128             return true;
129         }
130         // replaces multiple spaces with single space
131         // splits on the space after connect (gives us host and port)
132         // splits on : to get host as index 0 and port as index 1
133         String[] parts = text.trim().replaceAll(regex:" "+", replacement:" ").split(regex:" ")[1].split(regex:" :");
134         connect(parts[0].trim(), Integer.parseInt(parts[1].trim()));
135         return true;
136     } else if (isQuit(text)) {
137         sendDisconnect();
138         isRunning = false;
139         return true;
140     } else if (isName(text)) {
141         return true;
142     } else if (text.startsWith(prefix:"/joinroom")) {
143         String roomName = text.replace(target:"/joinroom", replacement:"").trim();
144         sendJoinRoom(roomName);
145         return true;
146     } else if (text.startsWith(prefix:"/createroom")) {
147         String roomName = text.replace(target:"/createroom", replacement:"").trim();
148         sendCreateRoom(roomName);
149     }

```

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#

```

IT114-450-Milestone.2 — java Project.server.Server — 98x30
at Project.server.Room.sendMessage(Room.java:93)
at Project.server.ServerThread.processPayload(ServerThread.java:186)
at Project.server.ServerThread.run(ServerThread.java:162)
Apr 04, 2024 1:14:04 AM Project.server.ServerThread run
INFO: Client disconnected
Apr 04, 2024 1:14:04 AM Project.server.ServerThread run
INFO: Exited thread loop, cleaning up connection
Apr 04, 2024 1:14:04 AM Project.server.ServerThread cleanup
INFO: Thread cleanup() start
Apr 04, 2024 1:14:04 AM Project.server.ServerThread cleanup
INFO: Thread cleanup() complete
Apr 04, 2024 1:14:16 AM Project.server.Server start
INFO: Waiting for next client
Apr 04, 2024 1:14:16 AM Project.server.Server start
INFO: Client connected
Apr 04, 2024 1:14:46 AM Project.server.ServerThread <init>
INFO: ServerThread created
Apr 04, 2024 1:14:46 AM Project.server.ServerThread run
INFO: Received from client: Type[CONNECT],ClientId[0,] ClientName[Jessica], Message[null]
Apr 04, 2024 1:14:46 AM Project.server.ServerThread send
INFO: Sent payloads: Type[CLIENT_ID],ClientId[2,] ClientName[null], Message[null]
Apr 04, 2024 1:14:46 AM Project.server.Server joinRoom
INFO: Client Jessica joining new room lobby
Apr 04, 2024 1:15:34 AM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[AdvancedProgramming], Message[/flip]
Apr 04, 2024 1:15:37 AM Project.server.ServerThread run
INFO: Received from client: Type[MESSAGE],ClientId[0,] ClientName[AdvancedProgramming], Message[/roll]

IT114-450-Milestone.2 — java Project.client.Client — 103x31
Closing input stream
Closing connection
Closed socket
/Roll@*
jessicadoodz@Jessicas-Air IT114-450-Milestone.2 ~ java Project.client.Client
Apr 04, 2024 1:14:15 AM Project.client.Client$1 run
INFO: Listening for input
Apr 04, 2024 1:14:15 AM Project.client.Client$1 run
INFO: Waiting for input
/names Jessica
Name set to Jessica
Apr 04, 2024 1:14:41 AM Project.client.Client$1 run
INFO: Waiting for input
/connect localhost:3001
Apr 04, 2024 1:14:46 AM Project.client.Client connect
INFO: Client connected
Apr 04, 2024 1:14:46 AM Project.client.Client$1 run
INFO: Waiting for input
Apr 04, 2024 1:14:46 AM Project.client.Client$2 run
INFO: Debug Info: Type[CLIENT_ID],ClientId[2,] ClientName[null], Message[null]
/names AdvancedProgramming
Name set to AdvancedProgramming
Apr 04, 2024 1:15:04 AM Project.client.Client$1 run
INFO: Waiting for input
/flip
Apr 04, 2024 1:15:34 AM Project.client.Client$1 run
INFO: Waiting for input
/roll
Apr 04, 2024 1:15:37 AM Project.client.Client$1 run
INFO: Waiting for input

```

## Message going out to all clients

### Checklist Items (1)

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

```

121    //UCID:jeo29
122    //Date:March 30,
123    private boolean processCommands(String message, ServerThread client) {
124        if (message.startsWith(prefix:"/roll")) {
125            String command = message.substring(beginIndex:6).trim();
126            try {
127                if (command.matches(regex:"\\d+d\\d+")) {
128                    // Format 2: /roll #d#
129                    String[] diceParts = command.split(regex:"d");
130                    int numberofDice = Integer.parseInt(diceParts[0].trim());
131                    int numberofSides = Integer.parseInt(diceParts[1].trim());
132                    StringBuilder result = new StringBuilder("Rolling " + numberofDice + "d" + numberofSides + ": ");
133                    for (int i = 0; i < numberofDice; i++) {
134                        int roll = (int) (Math.random() * numberofSides) + 1;
135                        result.append(roll);
136                        if (i < numberofDice - 1) {
137                            result.append(str:", ");
138                        }
139                    } <- #133-139 for (int i = 0; i < numberofDice; i++)
140                    sendMessage(client, result.toString());
141                } else {
142                    // Format 1: /roll 0 - X or 1 - X
143                    String[] rangeParts = command.split(regex:"-");
144                    int min = Integer.parseInt(rangeParts[0].trim());
145                    int max = Integer.parseInt(rangeParts[1].trim());
146                    int roll = (int) (Math.random() * (max - min + 1)) + min;
147                    sendMessage(client, "Rolling " + min + " - " + max + ": " + roll);
148                } <- #141-148 else
149                return true;
150            } catch (Exception e) {
151                return false;
152            }
153        } else if (message.equals(anObject:"/flip")) {
154            return handleFlipCommand(client);
155        } else if (message.startsWith(prefix:"/format")) {
156            String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
157            sendMessage(client, formattedMessage);
158            return true;
159        } <- #155-159 else if (message.startsWith("/format"))
160        return false;

```

## ServerThread

### Checklist Items (1)

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

```
171 //UCID:jeo29
172 //Date:March 30,
173 private String formatMessage(String message) {
174     // Bold text
175     message = message.replaceAll(regex:"\\*(.*?)\\*", replacement:"<b>$1</b>");
176     // Italic text
177     message = message.replaceAll(regex:"_(.*?)_", replacement:"<i>$1</i>");
178     // Red text
179     message = message.replaceAll(regex:"#r(.*)#", replacement:"<span style=\"color:red\">$1</span>");
180     // Green text
181     message = message.replaceAll(regex:"#g(.*)#", replacement:"<span style=\"color:green\">$1</span>");
182     // Blue text
183     message = message.replaceAll(regex:"#b(.*)#", replacement:"<span style=\"color:blue\">$1</span>");
184     // Underline text
185     message = message.replaceAll(regex:"#u(.*)#", replacement:"<u>$1</u>");
186     return message;
187 } <- #173-187 private String formatMessage(String message)
188
189
190
191     public void disconnectClient(ServerThread serverThread) {
192 }
193
Run | Debug
194 public static void main(String[] args) {
195     try (Room room = new Room(name:"Sample Room")) {
196         String sourceMessage = "Hello *bold* _italic_ #rgreen# #blue# #underline# text";
197         String transformedMessage = room.formatMessage(sourceMessage);
198
199         System.out.println("Source Message: " + sourceMessage);
200         System.out.println("Transformed Message: " + transformedMessage);
201     } <- #195-201 try (Room room = new Room("Sample Room"))
202 } <- #194-202 public static void main(String[] args)
203 } <- #18-203 public class Room implements AutoCloseable
204
```

## Room handling

### Checklist Items (1)

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

#### 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:

When a client sends a /roll command, the Room class distinguishes between dice rolls and random number ranges (e.g., 1-100) using the processCommands method. It then calculates the outcome based on the two possible outcomes. The result is the message that is sent between rooms. If sending fails due to a client disconnecting, the handleDisconnect method removes the client from the list. It keeps all users in rooms engaged.

#### Demonstrate Flip Command (1 pt.)

**^ COLLAPSE ^**

#### Task #1 - Points: 1

**^ COLLAPSE ^**

## **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**

```
121 //UCID:jeo29
122 //Date:March 30,
123 private boolean processCommands(String message, ServerThread client) {
124     if (message.startsWith(prefix:"/roll")) {
125         String command = message.substring(beginIndex:6).trim();
126         try {
127             if (command.matches(regex:"\\d+d\\d+")) {
128                 // Format 2: /roll #d#
129                 String[] diceParts = command.split(regex:"d");
130                 int numberofDice = Integer.parseInt(diceParts[0].trim());
131                 int numberofSides = Integer.parseInt(diceParts[1].trim());
132                 StringBuilder result = new StringBuilder("Rolling " + numberofDice + "d" + numberofSides + ": ");
133                 for (int i = 0; i < numberofDice; i++) {
134                     int roll = (int) (Math.random() * numberofSides) + 1;
135                     result.append(roll);
136                     if (i < numberofDice - 1) {
137                         result.append(str:", ");
138                     }
139                 } <- #133-139 for (int i = 0; i < numberofDice; i++)
140                 sendMessage(client, result.toString());
141             } else {
142                 // Format 1: /roll 0 - X or 1 - X
143                 String[] rangeParts = command.split(regex:"-");
144                 int min = Integer.parseInt(rangeParts[0].trim());
145                 int max = Integer.parseInt(rangeParts[1].trim());
146                 int roll = (int) (Math.random() * (max - min + 1)) + min;
147                 sendMessage(client, "Rolling " + min + " - " + max + ": " + roll);
148             } <- #141-148 else
149             return true;
150         } catch (Exception e) {
151             return false;
152         }
153     } else if (message.equals(anObject:"/flip")) {
154         return handleFlipCommand(client);
155     } else if (message.startsWith(prefix:"/format")) {
156         String formattedMessage = formatMessage(message.substring(beginIndex:8).trim());
157         sendMessage(client, formattedMessage);
158         return true;
159     } <- #155-159 else if (message.startsWith("/format"))
160     return false;
161 } <- #123-161 private boolean processCommands(String message, ServerThread ...
```

## Receiving the room

### **Checklist Items (1)**

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

```

.64 //Date:March 30,
.65     private boolean handleFlipCommand(ServerThread client) {
.66         int flip = (int) (Math.random() * 2);
.67         String result = (flip == 0) ? "Heads" : "Tails";
.68         sendMessage(client, "Coin flip: " + result);
.69         return true;
.70     } <- #165-170 private boolean handleFlipCommand(ServerThread client)
.71 ///UCID:jeo29
.72 //Date:March 30,
.73     private String formatMessage(String message) {
.74         // Bold text
.75         message = message.replaceAll(regex:"\\b+(.*?)\\b", replacement:"<b>$1</b>");
.76         // Italic text
.77         message = message.replaceAll(regex:"_(.*?)_", replacement:"<i>$1</i>");
.78         // Red text
.79         message = message.replaceAll(regex:"#r(.+?)#", replacement:"<span style=\"color:red\">$1</span>");
.80         // Green text
.81         message = message.replaceAll(regex:"#g(.+?)#", replacement:"<span style=\"color:green\">$1</span>");
.82         // Blue text
.83         message = message.replaceAll(regex:"#b(.+?)#", replacement:"<span style=\"color:blue\">$1</span>");
.84         // Underline text
.85         message = message.replaceAll(regex:"#u(.+?)#", replacement:"<u>$1</u>");
.86         return message;
.87     } <- #173-187 private String formatMessage(String message)
.88
.89
.90
.91     public void disconnectClient(ServerThread serverThread) {
.92     }
.93
Run | Debug
.94     public static void main(String[] args) {
.95         try (Room room = new Room(name:"Sample Room")) {
.96             String sourceMessage = "Hello #bold# _italic_ #rgreen# #bblue# #underline# text";
.97             String transformedMessage = room.formatMessage(sourceMessage);
.98
.99             System.out.println("Source Message: " + sourceMessage);
.100            System.out.println("Transformed Message: " + transformedMessage);
.101        } <- #195-201 try (Room room = new Room("Sample Room"))
.102    } <- #194-202 public static void main(String[] args)
.103 } <- #18-203 public class Room implements AutoCloseable

```

## Room handling the action

### Checklist Items (1)

#3 Room handling the flip action correctly

#### 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:

The flip command is handled by giving the output of either heads or tails. The message flows from client to room and is shared with all other users because the command is coded to identify the format and the output that is needed by providing the necessary results that will broadcast among all rooms. This allows for an interactive experience.

#### Demonstrate Formatted Messages (4 pts.)

**▲ COLLAPSE ▲**

#### Task #1 - Points: 1

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

### Details:

## 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:jeo29
//March 30,2024
private String formatMessage(String message) {
    // Bold text
    message = message.replaceAll(regex:"\\*(.*?)\\*", replacement:<b>$1</b>`);
    // Italic text
    message = message.replaceAll(regex:"_(.*?)_ ", replacement:<i>$1</i>`);
    // Red text
    message = message.replaceAll(regex:"#r(.*)#", replacement:<span style="color:red">$1</span>`);
    // Green text
    message = message.replaceAll(regex:"#g(.*)#", replacement:<span style="color:green">$1</span>`);
    // Blue text
    message = message.replaceAll(regex:"#b(.*)#", replacement:<span style="color:blue">$1</span>`);
    // Underline text
    message = message.replaceAll(regex:"#u(.*)#", replacement:<u>$1</u>`);
    return message;
} <- #173-187 private String formatMessage(String message)
```

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

### Checklist Items (1)

#1 Room code processing for bold

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

\_italicize

# with the specific letter for the color that it is assigned to is used

The logic works to convert the message to its final format. For example, #r would translate to the style color red. A simple markup of language then becoming visual style text.

Misc (1 pt.)

^ 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/jessicaodoom/jeo29-IT114-002/pull/17>

● [COLLAPSE ^](#)

## Task #2 - Points: 1

**Text:** Talk about any issues or learnings during this assignment

**Response:**

I had so many issues. I had to reach out to the professor so many times for assistance. I changed my mind about my project multiple times as well. One of my first issues was that I had made way too many changes so nothing was working. The professor then posted a new video which helped me start all over. Then I was having trouble compiling my files. I was running the wrong code and eventually figured that issue out as well. The professor was very helpful with his assistance and videos.

●

[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

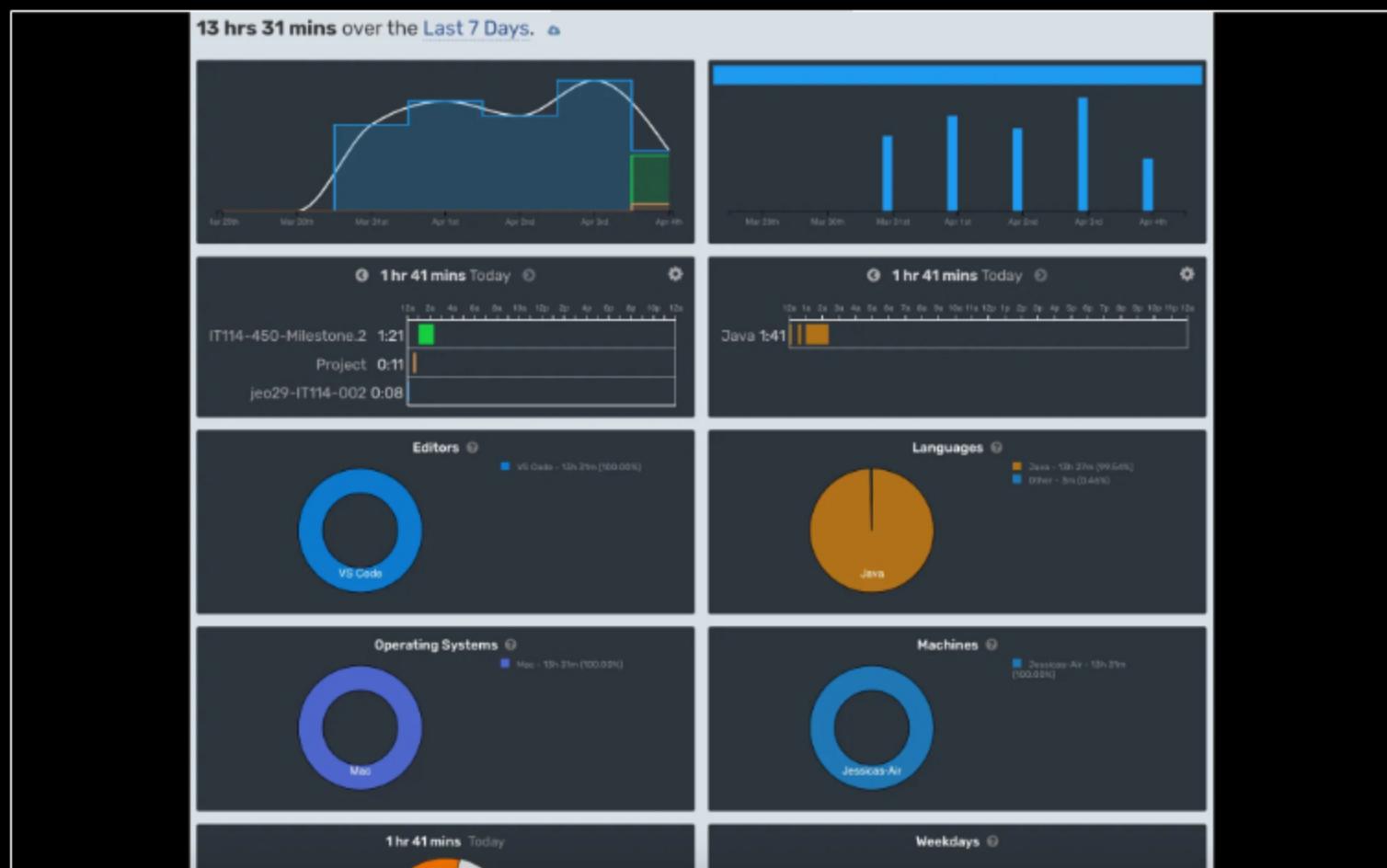
**Task Screenshots:**

### Gallery Style: Large View

Small

Medium

Large



End of Assignment