

# Submission Worksheet

**CLICK TO GRADE**

<https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-3-chatroom-2024-m24/grade/kh465>

Course: IT114-005-F2024

Assignment: [IT114] Milestone 3 Chatroom 2024 M24

Student: Keven H. (kh465)

## Submissions:

Submission Selection

1 Submission [submitted] 11/29/2024 6:38:55 PM

## Instructions

[^ COLLAPSE ^](#)

Implement the Milestone 3 features from the project's proposal document:

<https://docs.google.com/document/d/10NmveI97GTFPGfVwwQC96xSsobbSbk56145XizQG4/view>

Make sure you add your ucid/date as code comments where code changes are done All code changes should reach the Milestone3 branch Create a pull request from Milestone3 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: Milestone3

**Group**



Group: Basic UI

Tasks: 1

Points: 2

[^ COLLAPSE ^](#)

**Task**



Group: Basic UI

Task #1: UI Panels

Weight: ~100%

Points: ~2.00

## Details:

**All code screenshots must include ucid/date.**

**App screenshots must have the UCID in the title bar like the lesson gave.**

1

Columns: 1

### Sub-Task

## Group: Basic UI

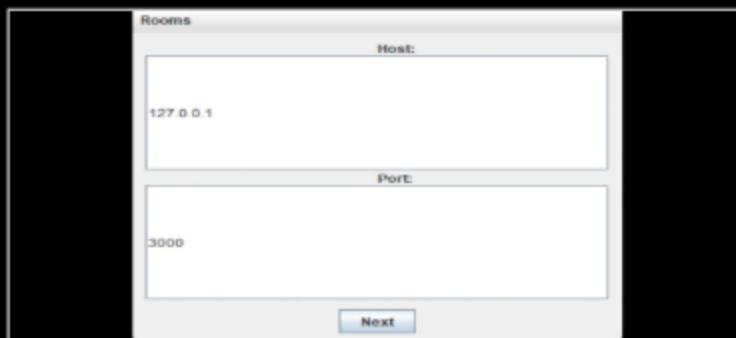
## Task #1: UI Panels

**Sub Task #1:** Show the ConnectionPanel by running the app (should have host/port)

## ▲ Task Screenshots

## Gallery Style: 2 Columns

4 2 1



## ConnectionPanel showing host and port

**Caption(s) (required)** ✓

**Caption Hint:** *Describe/highlight what's being shown*

### Sub-Task

## Group: Basic UI

## Task #1: UI Panels

Sub Task #2: Show the code related to the ConnectionPanel

## Task Screenshots

## Gallery Style: 2 Columns

4 2 1

```

// Add Bean Section
@UiBean("button")
UiButton button = new UiButton(text:"Next");
button.setAlignmentsX(2);
button.CMNTFR_ALIGNMENT); // Center the button
button.addActionListener((event) -> {
    SwingUtilities.invokeLater(() -> {
        boolean isValid = true;
        try {
            port = Integer.parseInt(portValue.getText());
            portError.setVisible(defFlagtrue); // Hide error label if valid
        } catch (NumberFormatException e) {
            portError.setVisible(true); // Show error label if invalid
            isValid = false;
        }
    });
    if (!isValid) {
        throw new NumberFormatException(e);
    }
    if (isValid) {
        host = hostValue.getText();
        construct.next(); // Navigate to the next card
    }
}); // < #02>/w swingutilities.invokeLater
content.add(button);
content.add(new CreateVerticalStrut(height:10)); // Add vertical spacing
content.add(button);

// Add the content panel in the center of the BorderLayout
this.add(content_panel, BorderLayout.CENTER);
this.setNames([CardName.COMPORT.name()]); // Set the name of the panel
content_panel.setNames([CardName.COMPORT.name(), $NAME]); // Add panel to controls
}, 1000);

```

### First part of the ConnectionPanel code

## Second part of the ConnectionPanel code

## Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

# Task Response Prompt

*Briefly explain how it works and how it's used*

Response:

ConnectionPanel takes a parameter of ICardControls, which creates various parts of the ConnectionPanel UI. It creates a host and port textbox, populates them with default values, and also creates errors but hides them unless it fails a boolean check.

### Sub-Task

Group: Basic UI

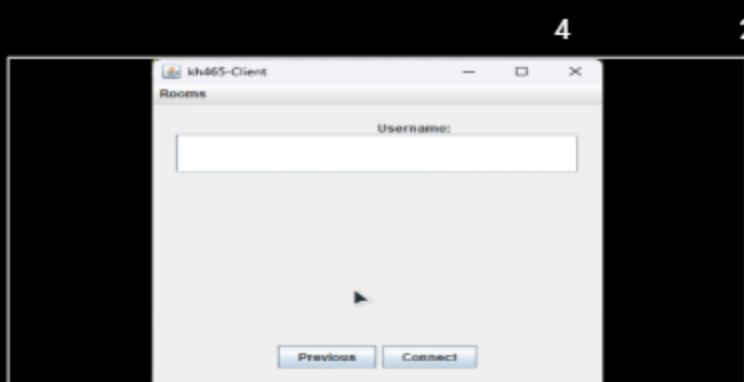
Task #1: UI Panels

Sub Task #3: show the UserDetailsPanel by running the app (should have username)



## Task Screenshots

Gallery Style: 2 Columns



UserDetailsPanel showing username

## Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

### Sub-Task

Group: Basic UI

Task #1: UI Panels

Sub Task #4: Show the code related to the UserDetailsPanel



## Task Screenshots

Gallery Style: 2 Columns

```
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.equals("connect")) {
        String host = hostField.getText();
        String port = portField.getText();
        String user = usernameField.getText();
        String pass = passwordField.getText();
        String[] args = {host, port, user, pass};
        ConnectionPanel cp = new ConnectionPanel(args);
        cp.setVisible(true);
    }
}
```

```
JPanels buttons = new JPanel();
buttons.add(previousButton);
buttons.add(connectButton);

content.add(Box.createVerticalGlue()); // Push buttons to the bottom
content.add(buttons);

// Add the content panel to the center of the BorderLayout
this.add(content, BorderLayout.CENTER);

// Add empty borders to the sides for spacing
this.setBorder(new EmptyBorder(10, left:10, bottom:10, right:10));

this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setTitle(CardView.USER_INFO.name());
controls.addPanel1(CardView.USER_INFO.name(), this);
}

<-- R24-78 public UserDetailsPanel(ICardControls controls)
```

**Caption(s) (required)** ✓Caption Hint: *Describe/highlight what's being shown***Task Response Prompt***Briefly explain how it works and how it's used*

Response:

UserDetailsPanel takes a parameter of ICardControls, which creates the UI for the username panel. It sets the layout, its alignment and its border. It creates a username textbox, waits for it to populate, and has a hidden error that only sets itself to be visible if the username field is left empty or if it contains a space

**Sub-Task**

Group: Basic UI

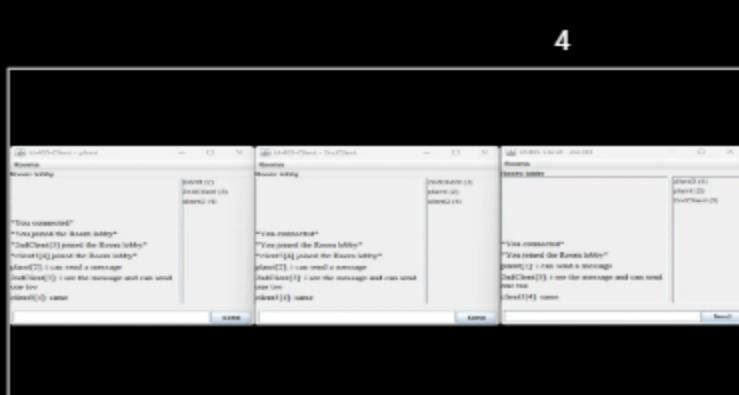


Task #1: UI Panels

Sub Task #5: Show the ChatPanel (there should be at least 3 users present and some example messages)

**Task Screenshots**

Gallery Style: 2 Columns



3 instances of ChatPanel alongside some example messages

**Caption(s) (required)** ✓Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Basic UI

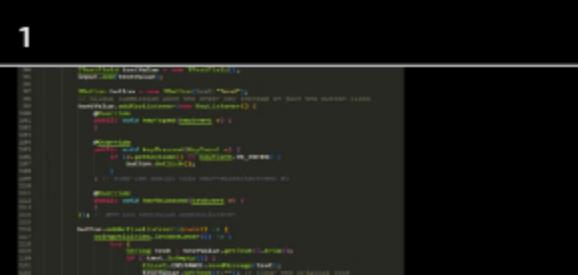
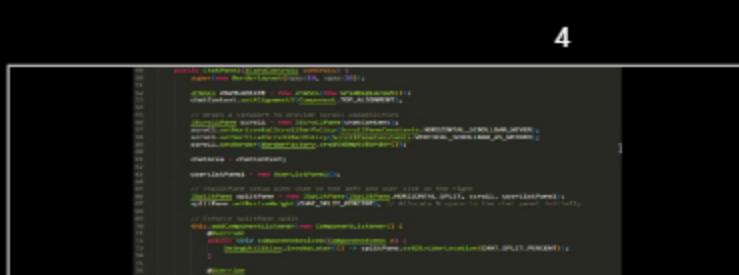


Task #1: UI Panels

Sub Task #6: Show the code related to the ChatPanel

**Task Screenshots**

Gallery Style: 2 Columns



## First part of the ChatPanel code

```
    33     @Override protected void onCreate(Bundle savedInstanceState) {  
    34         super.onCreate(savedInstanceState);  
    35         setContentView(R.layout.activity_main);  
    36     }  
    37  
    38     @Override public boolean onCreateOptionsMenu(Menu menu) {  
    39         getMenuInflater().inflate(R.menu.main, menu);  
    40         return true;  
    41     }  
    42  
    43     @Override public boolean onOptionsItemSelected(MenuItem item) {  
    44         int id = item.getItemId();  
    45         if (id == R.id.action_settings) {  
    46             Intent intent = new Intent(this, SettingsActivity.class);  
    47             startActivity(intent);  
    48             return true;  
    49         }  
    50         return super.onOptionsItemSelected(item);  
    51     }  
    52 }
```

## Second part of the ChatPanel code

```

140     public void componentAdded(ContainerEvent e) {
141         SwingUtilities.invokeLater( () -> {
142             if (chatArea.isVisible()) {
143                 chatArea.setExclude(true);
144                 chatArea.requestFocus();
145             }
146         });
147     } // <-- #140-146 SwingUtilities.invokeLaterLater
148
149     @Override
150     public void componentRemoved(ContainerEvent e) {
151         SwingUtilities.invokeLater( () -> {
152             if (chatArea.isVisible()) {
153                 chatArea.setExclude(false);
154                 chatArea.requestFocus();
155             }
156         });
157     } // <-- #151-157 SwingUtilities.invokeLaterLater
158 } // <-- #158-161 chatArea.addContainerListener
159
160 // ADD Vertical glue to push messages to the top
161 @Override
162 void addVerticalGlue() {
163     gbc.gridx = 0; // Column index 0
164     gbc.gridy = 0; // GridbagConstraints.RELATIVE; // Automatically move to the next row
165     gbc.weighty = 1.0f; // Give entire space vertically to this component
166     gbc.fill = GridBagConstraints.BOTH; // Fill both horizontally and vertically
167     chatArea.add(new VerticalGluePanel(), gbc);
168 }

```

### Third part of the ChatPanel code

**Caption(s) (required)** ✓

**Caption Hint:** *Describe/highlight what's being shown*

## Task Response Prompt

*Briefly explain how it works and how it's used (note the important parts of the ChatPanel)*

#### **Response:**

ChatPanel is created with ICardControls as an argument, creating chatContent where the messages are, along with a scroll that sets the scrollbar to appear whenever items overflow off screen. An input field is also created where the user can type their message out. This can be done through the send button, or through pressing the enter key. addActionListener is responsible for sending the message off and clearing the input field. addText adds user messages to the chat area, and scrolls the messages ascending

End of Task 1

### **End of Group: Basic UI**

Task Status: 1/1

## Group

## Group: Build-up

## Tasks: 2

**Points: 3**

COLLAPSE

## Task

## Group: Build-up

**Task #1: Results of /flip and /roll appear in a different format than regular chat text**

**Weight: ~50%**

Points: ~1.50

[COLLAPSE](#)

**i** Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

**Sub-Task**

100%

Group: Build-up

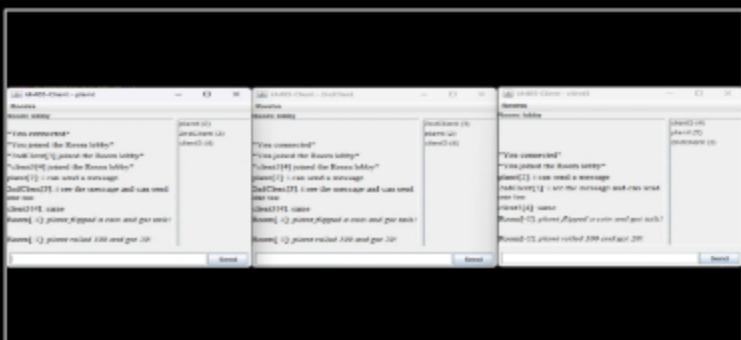
Task #1: Results of /flip and /roll appear in a different format than regular chat text

Sub Task #1: Show examples of it printing on screen

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



/roll and /flip italicised. All clients can see the format and message

**Caption(s) (required)** ✓

Caption Hint: *Describe/highlight what's being shown*

**Sub-Task**

100%

Group: Build-up

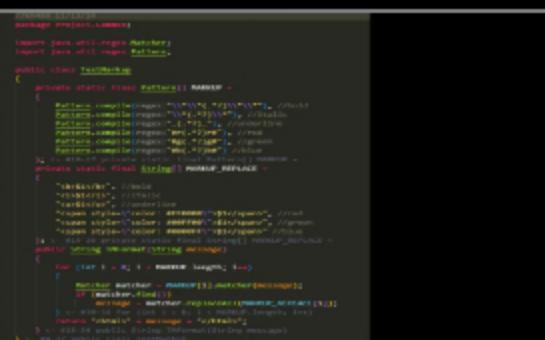
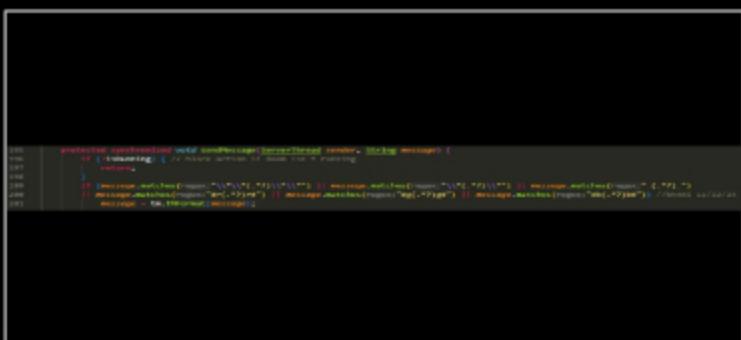
Task #1: Results of /flip and /roll appear in a different format than regular chat text

Sub Task #2: Show the code on the Room side that changes this format

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



Code in Room checking for formatting. Note that the transformations are done via TextMarkup. This calls the method on message.

TextMarkup showing patterns, their replacements, and the logic for replacing them

```

182     protected void handleText(ConversationThread render, TextMarkup textMarkup) {
183         if (textMarkup == null) {
184             return;
185         }
186         String text = textMarkup.getText();
187         int length = text.length();
188         int i = 0;
189         while (i < length) {
190             if (text.charAt(i) == '<') {
191                 int end = text.indexOf('>', i);
192                 if (end <= i + 1) {
193                     i = end;
194                 } else {
195                     String message = text.substring(i + 1, end);
196                     if (message.startsWith("bold")) {
197                         textMarkup.replaceText("b" + message.substring(5));
198                     } else if (message.startsWith("italic")) {
199                         textMarkup.replaceText("i" + message.substring(6));
200                     } else if (message.startsWith("underline")) {
201                         textMarkup.replaceText("u" + message.substring(8));
202                     } else if (message.startsWith("color")) {
203                         textMarkup.replaceText("color:" + message.substring(6));
204                     } else if (message.startsWith("bold italic")) {
205                         textMarkup.replaceText("b" + message.substring(8));
206                     } else if (message.startsWith("bold underline")) {
207                         textMarkup.replaceText("b" + message.substring(11));
208                     } else if (message.startsWith("bold color")) {
209                         textMarkup.replaceText("color:b" + message.substring(8));
210                     } else if (message.startsWith("italic underline")) {
211                         textMarkup.replaceText("i" + message.substring(11));
212                     } else if (message.startsWith("italic color")) {
213                         textMarkup.replaceText("color:i" + message.substring(8));
214                     } else if (message.startsWith("bold italic underline")) {
215                         textMarkup.replaceText("b" + message.substring(14));
216                     } else if (message.startsWith("bold italic color")) {
217                         textMarkup.replaceText("color:b" + message.substring(14));
218                     } else if (message.startsWith("bold underline color")) {
219                         textMarkup.replaceText("color:b" + message.substring(14));
220                     } else if (message.startsWith("italic underline color")) {
221                         textMarkup.replaceText("color:i" + message.substring(14));
222                     }
223                 }
224             }
225             i++;
226         }
227     }
228
229     protected void handleText(ConversationThread render) {
230         if (render == null) {
231             return;
232         }
233         String text = render.getText();
234         if (text != null) {
235             handleText(render, text);
236         }
237     }

```

handleRoll and handleFlip wrapping the messages in italics, which is handled by sendMessage

### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain what you did and how it works*

Response:

/roll, /flip and messages all call the sendMessage method, so message is checked for specific characters (e.g. \*\*\*) which then calls a method on TextMarkup (Room creates a TextMarkup object). TextMarkup has patterns for specific formats, and replacements for those formats. TMFormat loops through the MARKUP pattern array, and if it finds a match, it replaces those special characters with their HTML equivalents. handleRoll and handleFlip wrap the messages in HTML italics, so TMFormat does not need to replace it.

End of Task 1

### Task



Group: Build-up

Task #2: Text Formatting appears correctly on the UI

Weight: ~50%

Points: ~1.50

[▲ COLLAPSE ▲](#)

### Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

### Sub-Task

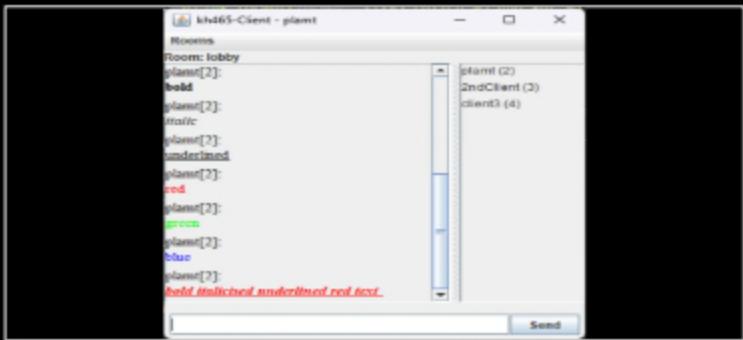
Group: Build-up

Task #2: Text Formatting appears correctly on the UI

Sub Task #1: Show examples of bold, italic, underline, each color implemented and a combination of bold, italic, underline, and one color in the same message

## Task Screenshots

Gallery Style: 2 Columns



All format variants shown individually, with a combination of all as the last message

#### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

##### Sub-Task



Group: Build-up

Task #2: Text Formatting appears correctly on the UI

Sub Task #2: Show the code changes necessary to get this to work

## ☒ Task Screenshots

Gallery Style: 2 Columns



Code changed from text/plain to text/html

#### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

## ≡ Task Response Prompt

*Briefly explain what was necessary and how it works*

Response:

JEditorPane needed to be changed from text/plain to text/html. Text/plain does not support HTML formatting, but text/html does. Wrapping the text in HTML formatting now displays the content as intended.

End of Task 2

End of Group: Build-up

Task Status: 2/2

Group



Group: New Features

Tasks: 2

Points: 4

[^ COLLAPSE ^](#)

### Task



Group: New Features

Task #1: Private messages via @username

Weight: ~50%

Points: ~2.00

[^ COLLAPSE ^](#)

#### i Details:

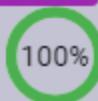
All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

#### Sub-Task



Group: New Features

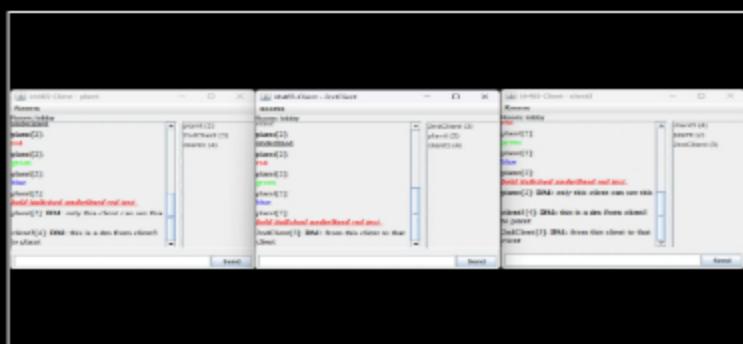
Task #1: Private messages via @username

Sub Task #1: Show a few examples across different clients (there should be at least 3 clients in the Room)

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



Three instances of clients DMing individual clients

#### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

#### Sub-Task

Group: New Features

Task #1: Private messages via @username

Sub Task #2: Show the client-side code that processes the text per the requirement



## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

374     public void sendMessage(String message) throws IOException {
375         if (processClientCommand(message)) {
376             return;
377         }
378         if (message.startsWith("@")) //kh465 11/26/24
379         {
380             sendDM(message);
381         }
382         else
383         {
384             Payload p = new Payload();
385             p.setPayloadType(PayloadType.MESSAGE);
386             p.setMessage(message);
387             send(p);
388         }
389     } <-- #326-389 public void sendMessage(String message) throws IOException

```

```

390     public void sendMessage(Message message, String recipientClientId) //kh465 11/26/24
391     {
392         String receivedMessage = message.replace("@", replacement);
393         String[] messageArray = receivedMessage.split(" ");
394         knownClients.replace(recipientClientId, receivedMessage);
395         if (recipientClientId.equals(messageArray[0]))
396         {
397             long recipientClientIdId = client.getClientId();
398             String formattedString = String.format("%s %s", "Recipient name and ID: ", recipientClientId);
399             messageArray[0] = formattedString;
400             message = String.join(" ", messageArray);
401             try {
402                 sendDM();
403             } catch (IOException e) {
404                 e.printStackTrace();
405             }
406         }
407         else {
408             System.out.println("No user found: " + recipientClientId);
409         }
410     }

```

sendMessage checking if message starts with @. If it does, sendDM. Detailed explanation explained below  
it's redirected to sendMessage

```

395     case PayloadType.DM:
396         processPayload(payload, getClientId(), payload.getMessage()); //kh465 11/26/24
397         break;
398

```

processPayload directing DM payload type to  
processMessage

**Caption(s) (required)** ✓

Caption Hint: *Describe/highlight what's being shown*

## ≡, Task Response Prompt

*Explain in concise steps how this logically works*

Response:

sendDM takes the message, replaces the @ with "", and is passed to a String array that separates by space and has a limit of 2. A lambda expression is then used to loop through each knownClient and if their name equals the first part of the String array, their getClientId method is called and stored in recipientClientId, which is added to a DMPayload and sent along to ServerThread.

**Sub-Task**

Group: New Features



Task #1: Private messages via @username

Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

## ▣ Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

128     public void processPayload(Payload payload, String clientId, String message) //kh465 11/26/24
129     {
130         DMPayload dmPayload = (DMPayload) payload;
131         currentRoom.sendMessage(dmPayload.getCurrentClientId(), dmPayload.getPayloadName());

```

**ServerThread extracts the senderId, recipientClientId and message and sends it along to Room**

**Caption(s) (required)** ✓

**Caption Hint:** *Describe/highlight what's being shown*

### **Task Response Prompt**

*Explain in concise steps how this logically works*

#### **Response:**

When ServerThread receives a PayloadType of DM, the DM case is used, casting the current payload into a DMPayload, and calling the currentRoom.sendDM method with the senderId, recipientClientId and message extracted from the payload and sent along to the Room

### Sub-Task

## Group: New Features

100%

Task #1: Private messages via @username

Sub Task #4: Show the Room code that verifies the id and sends the message to both the sender and receiver

## Task Screenshots

## Gallery Style: 2 Columns

4 2 1

```
223     private void sendMessage(Sendable sender, long recipientId, String message) //throws IOException
224     {
225         long senderId = sender.getSenderId();
226
227         Client client = backlog.get((Object) client);
228         if(client == null) return;
229
230         client.sendMessage(senderId, "client <--> " + message);
231         if(client.getSenderId() == (senderId))
232             client.sendMessage(senderId, "client <--> " + message);
233         else
234             sender.sendMessage(senderId, "client <--> " + message);
235
236     } // x222.138 ClientDefinition-forBack
237
238     @Override protected synchronized void sendAll(Sendable sender, long ...
```

**sendDM. Further explanation below**

**Caption(s) (required)** ✓

**Caption Hint:** *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

### **Response:**

`sendDM` uses a lambda expression for each `clientsInRoom`, checking their `clientId` against the `recipientClientId`. If it matches, that clients' `sendMessage` method gets called with the `senderId` and their message, appended with `DM:` to denote that it's a DM from the specified client. This check is also done for the `senderId` and the same thing is done for clarity, otherwise the sender would not see their own message.

### Task



Group: New Features  
Task #2: Mute and Unmute  
Weight: ~50%  
Points: ~2.00

[▲ COLLAPSE ▲](#)

#### ① Details:

All code screenshots must include ucid/date.  
App screenshots must have the UCID in the title bar like the lesson gave.



- Client-side will implement a /mute and /unmute command (i.e., /mute Bob or /unmute Bob)

Columns: 1

#### Sub-Task



Group: New Features  
Task #2: Mute and Unmute  
Sub Task #1: Show a few examples across different clients (there should be at least 3 clients in the Room)

## ■ Task Screenshots

Gallery Style: 2 Columns

4      2      1

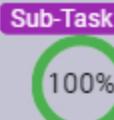


Missing Caption

#### Caption(s) (required)

Caption Hint: *Describe/highlight what's being shown*

Missing caption(s)



Group: New Features  
Task #2: Mute and Unmute  
Sub Task #2: Show the client-side code that processes the text per the requirement

## ■ Task Screenshots

Gallery Style: 2 Columns

4      2      1

```

267     case MUTE: //kh465 11/26/24
268         KnownClients.forEach((id, client) -> {
269             if (client.getNickname().equals(commandValue)) {
270                 try {
271                     sendMute(commandValue);
272                 } catch (Exception e) {
273                     logger.error("Error sending mute to client " + id);
274                 }
275             }
276         });
277         INSTANCE.warning(String.format("No user found! %s", commandValue));
278     }
279     case UNMUTE: //kh465 11/26/24
280         KnownClients.forEach((id, client) -> {
281             if (client.getNickname().equals(commandValue)) {
282                 try {
283                     sendUnmute(id);
284                 } catch (Exception e) {
285                     logger.error("Error unmuting client " + id);
286                 }
287             }
288         });
289     }
290     case SENDCOMMAND: //kh465 11/26/24
291     }
292 }
293 }
```

processClientCommand. Further explanation below

sendMute assigning mutedClientId from given client name

#### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

Using processClientCommand, the commandValue gets passed to the method sendMute. sendMute creates a new DMPayload, creates a long called mutedClientId which is assigned to the first entry found that matches the name passed along, sets the recClientId to the mutedClientId, sets the PayloadType as MUTE, and sends it to ServerThread. **Note:** I did not create a sendUnmute method since I could not create logic to actually mute a client. Sorry :(

#### Sub-Task

Group: New Features

100%

Task #2: Mute and Unmute

Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

<pre> 3.32 3.33     case MUTE: //kh465 11/26/24 3.34         DMPayload mutePayload = (DMPayload) payload; 3.35         currentRoom.handleMute(this, mutePayload.getRecClientId());         break;     } } </pre>	<pre> 136 137     case UNMUTE: //kh465 11/26/24 138         DMPayload unmutePayload = (DMPayload) payload; 139         currentRoom.handleUnmute(this, unmutePayload.getRecClientId());         break;     } } </pre>
--	--

PayloadType MUTE passing its data to Room

PayloadType UNMUTE passing its data to Room

#### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

payload is casted to DMPayload, which calls the handleMute method in currentRoom and passes the senderId alongside the recClientId (mutedClientId). The same code is used for processPayload case UNMUTE

**Sub-Task**

100%

Group: New Features

Task #2: Mute and Unmute

Sub Task #4: Show the Room code that verifies the id and add/removes the muted name to/from the ServerThread's list

**Task Screenshots**

Gallery Style: 2 Columns

4      2      1

```

242     protected void handleMute(ServerThread sender, Long clientToMute) //kn405 11/26/24
243     {
244         clientsInRoom.forEach((id, client) -> {
245             if(client.getClientId() == clientToMute)
246             {
247                 sender.addToMuteList(clientToMute);
248             }
249         });
250     }
251
252     ● protected void handleUnmute(ServerThread sender, Long clientToUnmute) //kn405 11/26/24
253     {
254         clientsInRoom.forEach((id, client) -> {
255             if(client.getClientId() == clientToUnmute)
256             {
257                 sender.removeFromMuteList(clientToUnmute);
258             }
259         });
260     }

```

handleMute and handleUnmute calling methods from ServerThread

```

151     public void addToMuteList(Long clientIdToMute) //kn405 11/26/24
152     {
153         mutedClients.add(clientIdToMute);
154     }
155
156     public void removeFromMuteList(Long clientIdToUnmute) //kn405 11/26/24
157     {
158         mutedClients.remove(clientIdToUnmute);
159     }
160
161     public ArrayList<Long> getMuteList() //kn405 11/26/24
162     {
163         return mutedClients;
164     }
165
166     public boolean isMuted(Long mutedClientId) //kn405 11/26/24
167     {
168         return mutedClients.contains(mutedClientId);
169     }
170
171     public boolean sendMute(Long clientId, Long mutedClientId) //kn405 11/26/24
172     {
173         CMPPayload mp = new CMPPayload();
174         mp.setRecClientId(mutedClientId);
175         mp.setClientID(clientId);
176         mp.setClientType("client");
177         return send(mp);
178     }
179 } <- m27-27 public boolean sendMute(Long clientId, Long mutedClientId) ...

```

Code in ServerThread relating to the mute list

**Caption(s) (required)** ✓

Caption Hint: *Describe/highlight what's being shown*

**Task Response Prompt**

*Explain in concise steps how this logically works*

Response:

processPayload in ServerThread extracts the senderId and recClientId for both MUTE and UNMUTE and sends them to their respective methods. In Room, handleMute loops through each clientsInRoom and if their getClientId matches the long clientToMute, it calls that ServerThreads' addToMuteList method. Similar code is used for handleUnmute, except it removes clientToUnmute via removeFromMuteList

**Sub-Task**

Group: New Features

0%

Task #2: Mute and Unmute

Sub Task #5: Show the Room code that checks the mute list during send message, private message, and any other relevant location

**Task Screenshots**

Gallery Style: 2 Columns

4      2      1



Missing Caption

Missing Caption

### Caption(s) (required)

Caption Hint: *Describe/highlight what's being shown*

Missing caption(s)

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

Missing Response

### Sub-Task

0%

Group: New Features

Task #2: Mute and Unmute

Sub Task #6: Show terminal supplemental evidence per the requirements (refer to the details of this task)

## Task Screenshots

Gallery Style: 2 Columns

4

2

1



Missing Caption

### Caption(s) (required)

Caption Hint: *Describe/highlight what's being shown*

Missing caption(s)

End of Task 2

End of Group: New Features

Task Status: 1/2

### Group

100%

Group: Misc

Tasks: 3

Points: 1

▲ COLLAPSE ▲

### Task

100%

Group: Misc

Task #1: Add the pull request link for the branch

Weight: ~33%

[▲ COLLAPSE ▲](#)**i Details:**

Note: the link should end with /pull/#

**↪ Task URLs**

URL #1

<https://github.com/kh465/kh465-IT114-005/pull/14>

URL

<https://github.com/kh465/kh465-IT114-005/pull/>

End of Task 1

**Task**

Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)**≡ Task Response Prompt**

Response:

The biggest issues I encountered during the assignment was the creation of DMs and muting/unmuting users. I was able to overcome DMing users but I was unable to figure out how to create logic to actually mute and unmute users.

End of Task 2

**Task**

Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)**i 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: 2 Columns

4 2 1



WakaTime showing the hours spent over the last 7 days

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment