Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-006-S2024/it114-chatroom-milestone-3-2024/grade/bm47

IT114-006-S2024 - [IT114] Chatroom Milestone 3 2024

Submissions:

Submission Selection

1 Submission [active] 4/11/2024 3:08:50 PM

Instructions

COLLAPSE

Implement the Milestone 3 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145Xi
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

Tasks: 14 Points: 10.00

● Basic UI (2 pts.)

COLLAPSE

● COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the following

Checklist *The checkboxes are for your own tracking

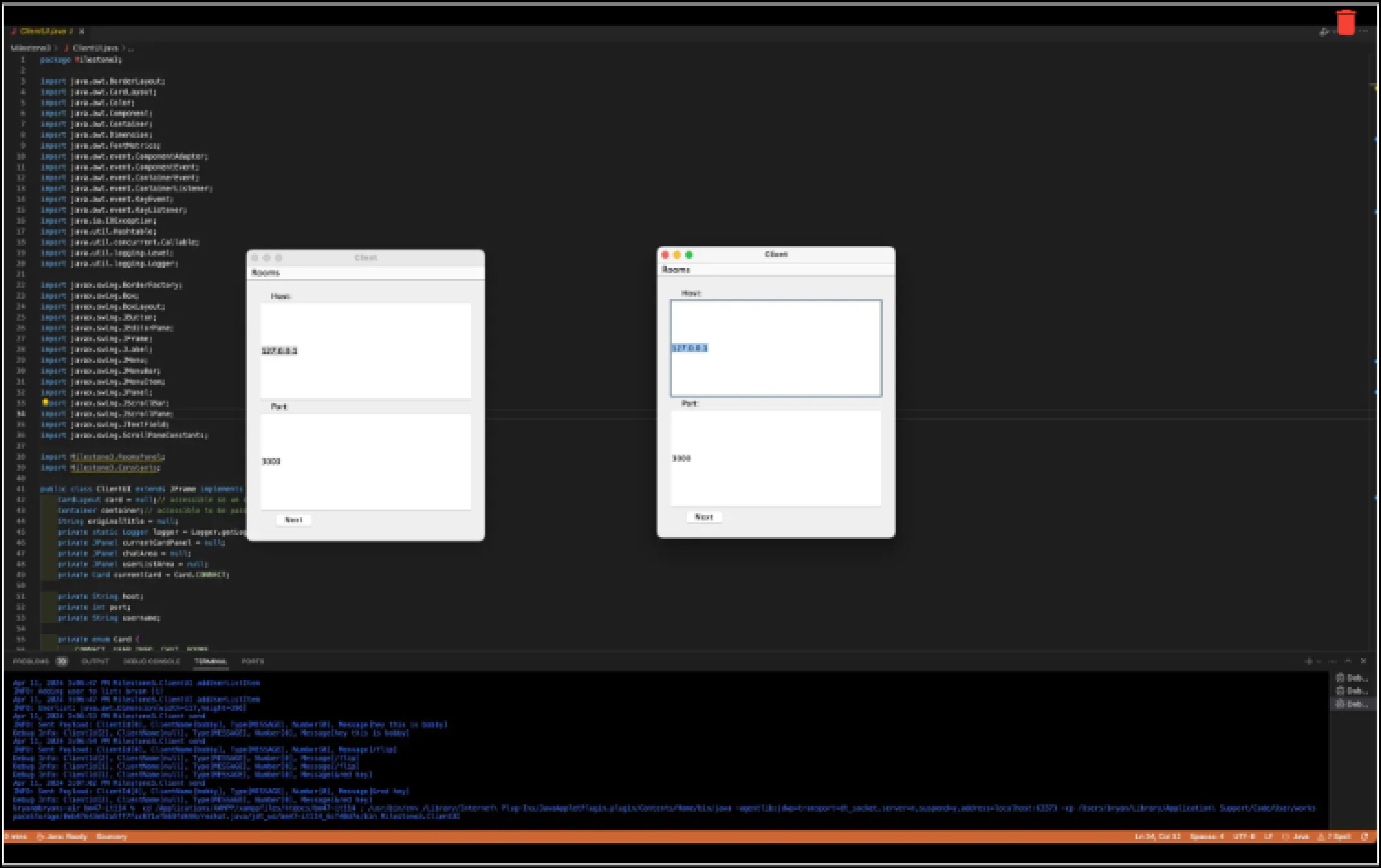
#	Points	Details

#1	1	Connection Panel
#2	1	User Details Panel
#3	1	Chat Panel
#4	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

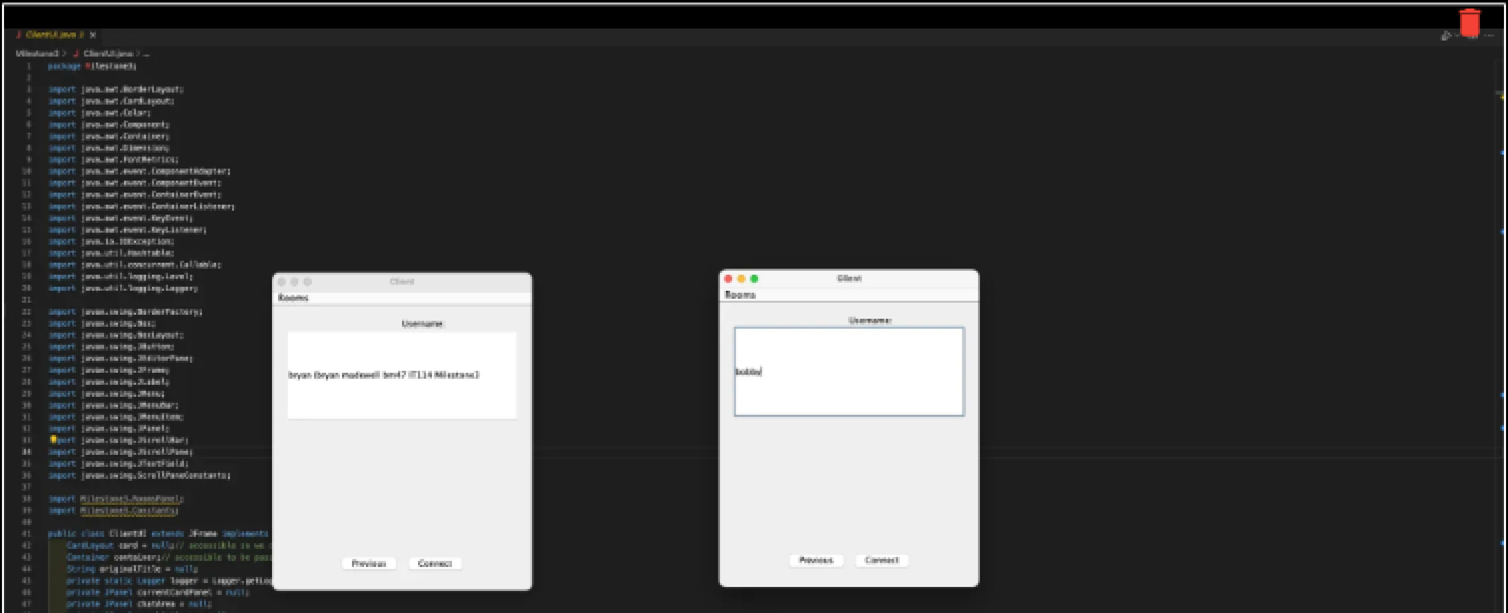

ClientUI.java 2
Milestone3 > ClientUI.java > ...
Client
Rooms
Host:
127.0.0.1
Port:
3000
Next
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
package Milestone3;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ContainerEvent;
import java.awt.event.ContainerListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;
import java.util.Hashtable;
import java.util.concurrent.Callable;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;

import Milestone3.RoomsPanel;
import Milestone3.Constants;

public class ClientUI extends JFrame implements
    CardLayout card = null;// accessible so we c
    Container container;// accessible to be pas
    String originalTitle = null;
    private static Logger logger = Logger.getLo
    private JPanel currentCardPanel = null;
    private JPanel chatArea = null;
    private JPanel userListArea = null;
    private Card currentCard = Card.CONNECT;

    private String host;
    private int port;
    private String username;

    private enum Card {
        CONNECT, USER_INFO, CHAT, ROOMS
```

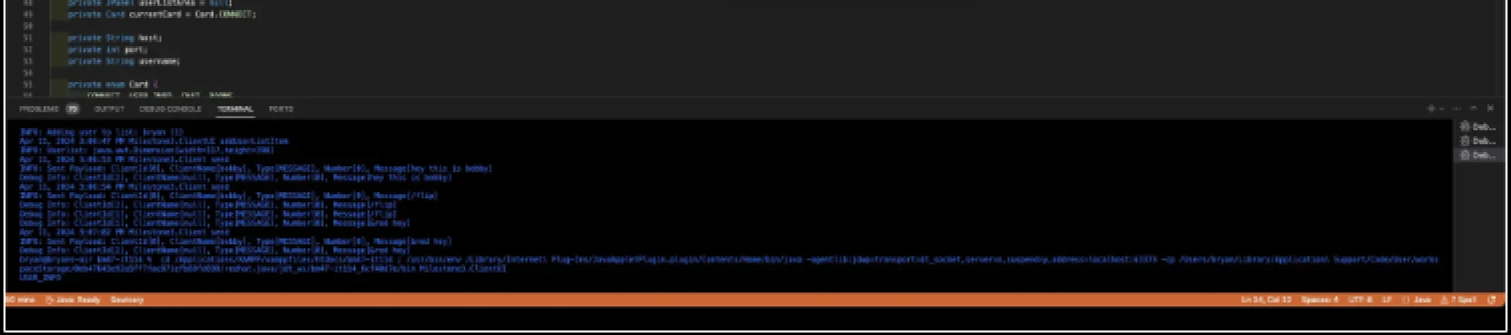
```
Apr 11, 2024 3:06:47 PM Milestone3.ClientUI addUserListItem
INFO: Adding user to list: bryan [1]
Apr 11, 2024 3:06:47 PM Milestone3.ClientUI addUserListItem
INFO: UserList: java.awt.Dimension[width=117,height=396]
Apr 11, 2024 3:06:53 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[hey this is bobby]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[hey this is bobby]
Apr 11, 2024 3:06:54 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[1], ClientName[null], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[1], ClientName[null], Type[MESSAGE], Number[0], Message[&red hey]
Apr 11, 2024 3:07:02 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[&red hey]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[&red hey]
bryan@bryans-air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdocs/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:63373 -cp /Users/bryan/Library/Application\ Support/Code/User/works
paceStorage/[illegible]/redhat.java/jdt_ws/bm47-it114_[illegible]/bin Milestone3.ClientUI
```

connection panel for 2 users of the chatroom

Checklist Items (0)


ClientUI.java 2
Milestone3 > ClientUI.java > ...
Client
Rooms
Username
bryan (bryan madawell bm47 IT114 Milestone3)
Username:
bobby
Previous
Connect

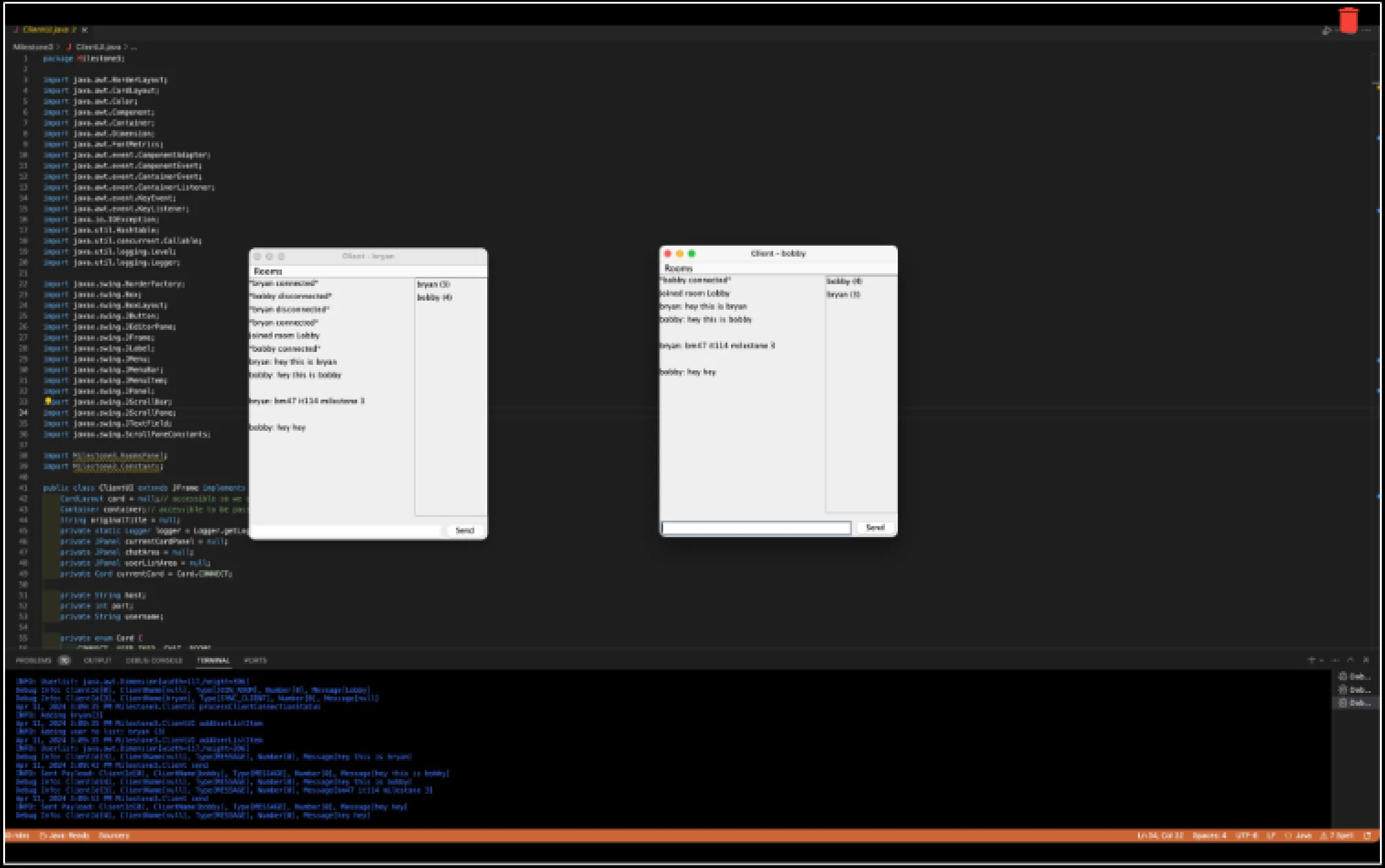
```
package Milestone3;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ContainerEvent;
import java.awt.event.ContainerListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;
import java.util.Hashtable;
import java.util.concurrent.Callable;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;

import Milestone3.RoomsPanel;
import Milestone3.Constants;

public class ClientUI extends JFrame implements
    CardLayout card = null;// accessible so we c
    Container container;// accessible to be pas
    String originalTitle = null;
    private static Logger logger = Logger.getLo
    private JPanel currentCardPanel = null;
    private JPanel chatArea = null;
```


```
    private JPanel userListArea = null;
    private Card currentCard = Card.CONNECT;

    private String host;
    private int port;
    private String username;

    private enum Card {
        CONNECT, USER_INFO, CHAT, ROOMS
```

```
INFO: Adding user to list: bryan (3)
Apr 11, 2024 3:06:47 PM Milestone3.ClientUI addUserListItem
INFO: Userlist: java.awt.Dimension[width=117,height=306]
Apr 11, 2024 3:06:53 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[hey this is bobby]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[hey this is bobby]
Apr 11, 2024 3:06:54 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[/flip]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[bred hey]
Apr 11, 2024 3:07:02 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[bred hey]
Debug Info: ClientId[2], ClientName[null], Type[MESSAGE], Number[0], Message[bred hey]
bryan@bryans-air bm47-it114 %  cd /Applications/XAMPP/xamppfiles/htdocs/bm47-it114 ; /usr/bin/env /Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:61576 -cp /Users/bryan/Library/Application\ Support/Code/User/workspaceStorage/[illegible]/redhat.java/jdt_ws/bm47-it114_[illegible]/bin Milestone3.ClientUI
USER_INFO
```

entering the usernames (i put my UCID and the class just for proof)

Checklist Items (0)


```
package Milestone3;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ContainerEvent;
import java.awt.event.ContainerListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;
import java.util.Hashtable;
import java.util.concurrent.Callable;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ScrollPaneConstants;

import Milestone3.RoomsPanel;
import Milestone3.Constants;

public class ClientUI extends JFrame implements
    CardLayout card = null;// accessible so we c
    Container container;// accessible to be pass
    String originalTitle = null;
    private static Logger logger = Logger.getLo
    private JPanel currentCardPanel = null;
    private JPanel chatArea = null;
    private JPanel userListArea = null;
    private Card currentCard = Card.CONNECT;

    private String host;
    private int port;
    private String username;

    private enum Card {
        CONNECT, USER_INFO, CHAT, ROOMS
```

Client - bryan

Rooms

```
*bryan connected*
*bobby disconnected*
*bryan disconnected*
*bryan connected*
Joined room Lobby
*bobby connected*
bryan: hey this is bryan
bobby: hey this is bobby

bryan: bm47 it114 milestone 3

bobby: hey hey
```

bryan (3)
bobby (4)

Client - bobby

Rooms

```
*bobby connected*
Joined room Lobby
bryan: hey this is bryan
bobby: hey this is bobby

bryan: bm47 it114 milestone 3

bobby: hey hey
```

bobby (4)
bryan (3)

```
INFO: Userlist: java.awt.Dimension[width=117,height=306]
Debug Info: ClientId[0], ClientName[null], Type[JOIN_ROOM], Number[0], Message[Lobby]
Debug Info: ClientId[3], ClientName[bryan], Type[SYNC_CLIENT], Number[0], Message[null]
Apr 11, 2024 3:05:35 PM Milestone3.ClientUI processClientConnectionStatus
INFO: Adding bryan[3]
Apr 11, 2024 3:05:35 PM Milestone3.ClientUI addUserListItem
INFO: Adding user to list: bryan (3)
Apr 11, 2024 3:05:35 PM Milestone3.ClientUI addUserListItem
INFO: Userlist: java.awt.Dimension[width=117,height=306]
Debug Info: ClientId[3], ClientName[null], Type[MESSAGE], Number[0], Message[hey this is bryan]
Apr 11, 2024 3:05:42 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[hey this is bobby]
Debug Info: ClientId[4], ClientName[null], Type[MESSAGE], Number[0], Message[hey this is bobby]
Debug Info: ClientId[3], ClientName[null], Type[MESSAGE], Number[0], Message[bm47 it114 milestone 3]
Apr 11, 2024 3:05:53 PM Milestone3.Client send
INFO: Sent Payload: ClientId[0], ClientName[bobby], Type[MESSAGE], Number[0], Message[hey hey]
Debug Info: ClientId[4], ClientName[null], Type[MESSAGE], Number[0], Message[hey hey]
```

here is the chatroom working with the chat panels in view

Checklist Items (0)

● Formatting (2 pts.)

^COLLAPSE ^

●

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots demoing flip and roll commands

Checklist *The checkboxes are for your own tracking

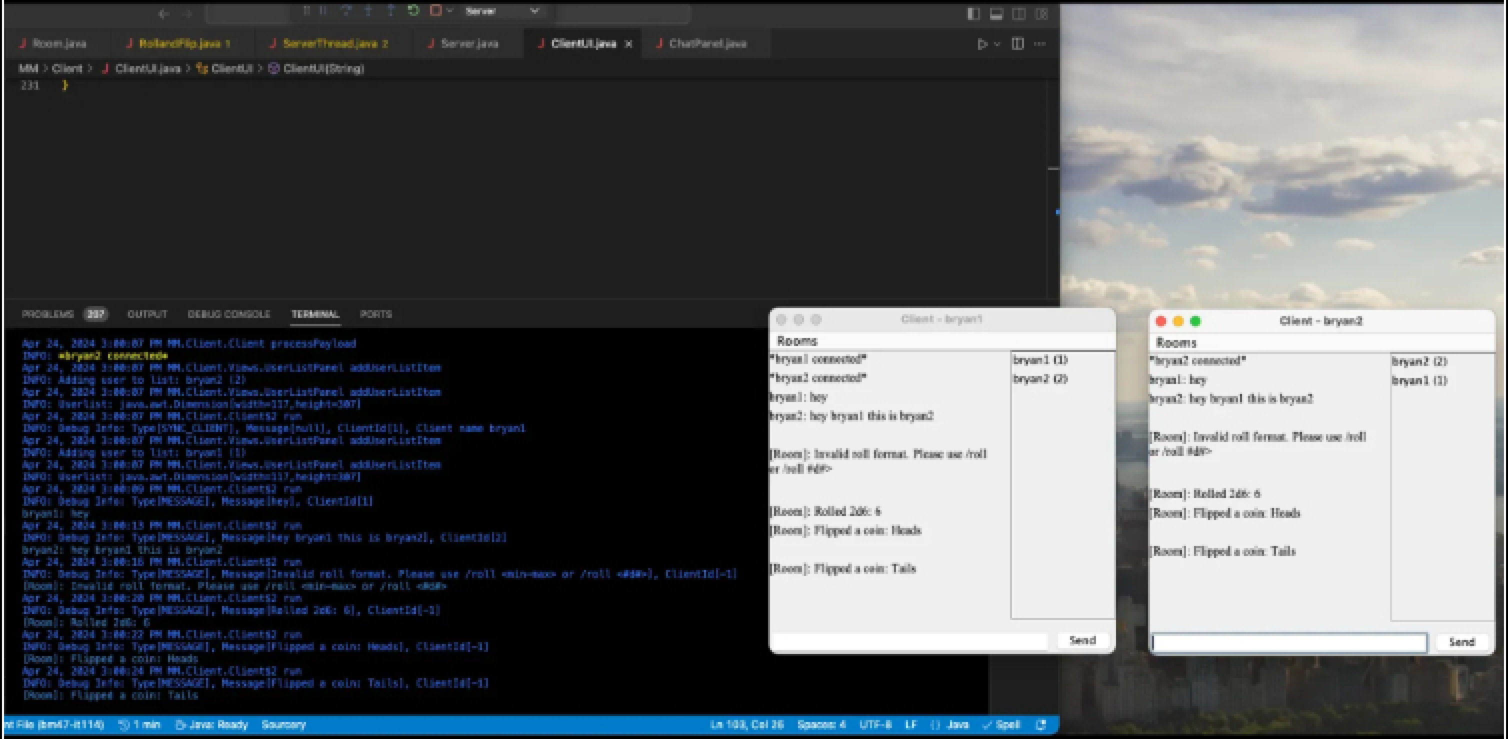
#	Points	Details
■ #1	1	Flip output in a different format than normal messages

🟦 #2	1	Roll # output in a different format than normal messages
🟦 #3	1	Roll #d# output in a different format than normal messages
🟦 #4	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
Apr 24, 2024 3:00:07 PM MM.Client.Client processPayload
INFO: *bryan2 connected*
Apr 24, 2024 3:00:07 PM MM.Client.Views.UserListPanel addUserListItem
INFO: Adding user to list: bryan2 (2)
Apr 24, 2024 3:00:07 PM MM.Client.Views.UserListPanel addUserListItem
INFO: Userlist: java.awt.Dimension[width=117,height=307]
Apr 24, 2024 3:00:07 PM MM.Client.Client$2 run
INFO: Debug Info: Type[SYNC_CLIENT], Message[null], ClientId[1], Client name bryan1
Apr 24, 2024 3:00:07 PM MM.Client.Views.UserListPanel addUserListItem
INFO: Adding user to list: bryan1 (1)
Apr 24, 2024 3:00:07 PM MM.Client.Views.UserListPanel addUserListItem
INFO: Userlist: java.awt.Dimension[width=117,height=307]
Apr 24, 2024 3:00:09 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[hey], ClientId[1]
bryan1: hey
Apr 24, 2024 3:00:13 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[hey bryan1 this is bryan2], ClientId[2]
bryan2: hey bryan1 this is bryan2
Apr 24, 2024 3:00:16 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Invalid roll format. Please use /roll <min-max> or /roll <#d#>], ClientId[-1]
[Room]: Invalid roll format. Please use /roll <min-max> or /roll <#d#>
Apr 24, 2024 3:00:20 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Rolled 2d6: 6], ClientId[-1]
[Room]: Rolled 2d6: 6
Apr 24, 2024 3:00:22 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: Heads], ClientId[-1]
[Room]: Flipped a coin: Heads
Apr 24, 2024 3:00:24 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: Tails], ClientId[-1]
[Room]: Flipped a coin: Tails
```

Client - bryan1:

```
*bryan1 connected*
*bryan2 connected*
bryan1: hey
bryan2: hey bryan1 this is bryan2

[Room]: Invalid roll format. Please use /roll
or /roll #d#>

[Room]: Rolled 2d6: 6
[Room]: Flipped a coin: Heads

[Room]: Flipped a coin: Tails
```

bryan1 (1)
bryan2 (2)

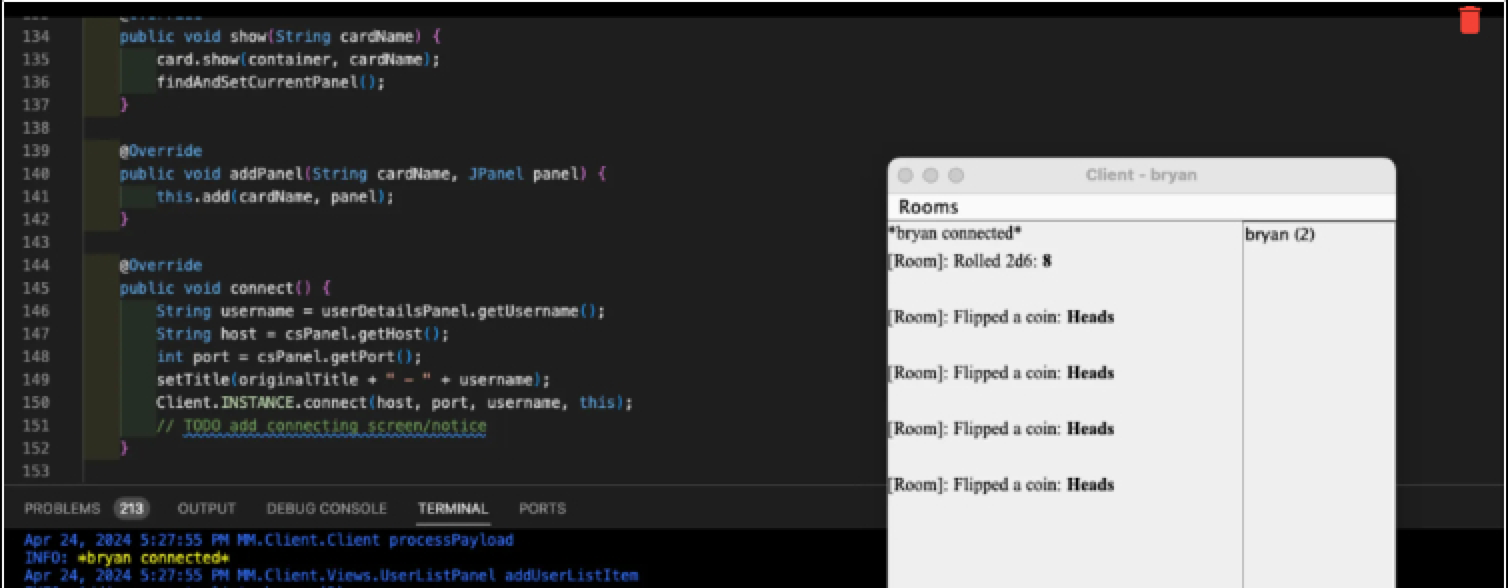
Client - bryan2:

```
*bryan2 connected*
bryan1: hey
bryan2: hey bryan1 this is bryan2

[Room]: Invalid roll format. Please use /roll
or /roll #d#>

[Room]: Rolled 2d6: 6
[Room]: Flipped a coin: Heads

[Room]: Flipped a coin: Tails
```

bryan2 (2)
bryan1 (1)

Attached is a screenshot of the UI working. Also, I first used the /roll command incorrectly so it will show the correct usage. I then did "/roll 2d6" and then showed the /flip command working twice and getting a successful output all times.

Checklist Items (0)



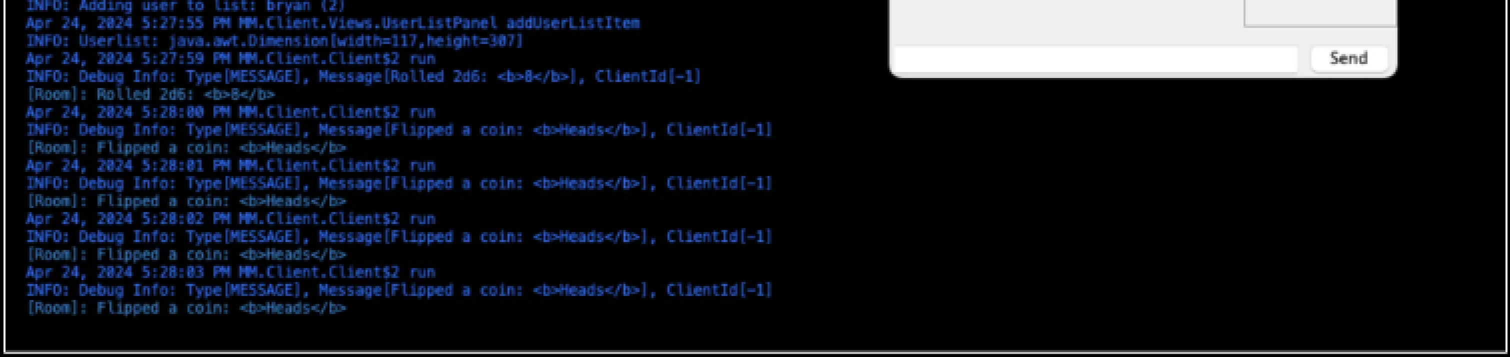
```java
    public void show(String cardName) {
        card.show(container, cardName);
        findAndSetCurrentPanel();
    }

    @Override
    public void addPanel(String cardName, JPanel panel) {
        this.add(cardName, panel);
    }

    @Override
    public void connect() {
        String username = userDetailsPanel.getUsername();
        String host = csPanel.getHost();
        int port = csPanel.getPort();
        setTitle(originalTitle + " - " + username);
        Client.INSTANCE.connect(host, port, username, this);
        // TODO add connecting screen/notice
    }
```

```
Apr 24, 2024 5:27:55 PM MM.Client.Client processPayload
INFO: *bryan connected*
Apr 24, 2024 5:27:55 PM MM.Client.Views.UserListPanel addUserListItem
```

Client - bryan:

```
*bryan connected*
[Room]: Rolled 2d6: **8**

[Room]: Flipped a coin: **Heads**

[Room]: Flipped a coin: **Heads**

[Room]: Flipped a coin: **Heads**

[Room]: Flipped a coin: **Heads**
```

bryan (2)



```
INFO: Adding user to list: bryan (2)
Apr 24, 2024 5:27:55 PM MM.Client.Views.UserListPanel addUserListItem
INFO: Userlist: java.awt.Dimension[width=117,height=307]
Apr 24, 2024 5:27:59 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Rolled 2d6: <b>8</b>], ClientId[-1]
[Room]: Rolled 2d6: <b>8</b>
Apr 24, 2024 5:28:00 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: <b>Heads</b>], ClientId[-1]
[Room]: Flipped a coin: <b>Heads</b>
Apr 24, 2024 5:28:01 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: <b>Heads</b>], ClientId[-1]
[Room]: Flipped a coin: <b>Heads</b>
Apr 24, 2024 5:28:02 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: <b>Heads</b>], ClientId[-1]
[Room]: Flipped a coin: <b>Heads</b>
Apr 24, 2024 5:28:03 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[Flipped a coin: <b>Heads</b>], ClientId[-1]
[Room]: Flipped a coin: <b>Heads</b>
```

note: i just saw that the outputs are supposed to be in a different format than normal messages, so I made the output bold by adding HTML tags as you can see in the terminal.

Checklist Items (0)

●

^COLLAPSE ^

Task #2 - Points: 1

Text: Screenshots demoing custom text formatting

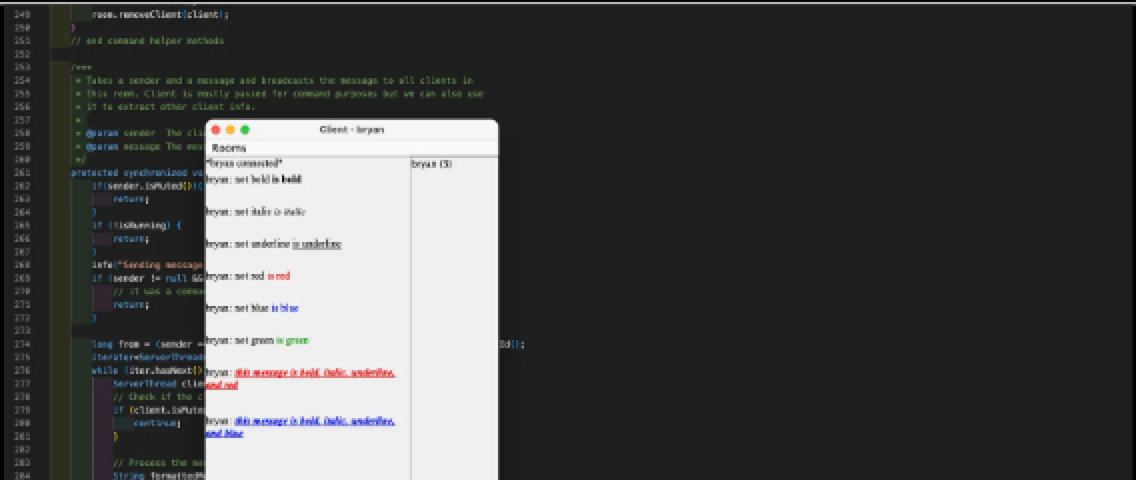
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Custom text formatting for bold working (Part of the message should appear bold)
#2	1	Custom text formatting for italic working (Part of the message should appear italic)
#3	1	Custom text formatting for underline working (Part of the message should appear underline)
#4	1	Custom text formatting for red working (Part of the message should appear red)
#5	1	Custom text formatting for blue working (Part of the message should appear blue)
#6	1	Custom text formatting for green working (Part of the message should appear green)
#7	1	Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should have all 4 formats applied at once)
#8	1	Clearly caption screenshots

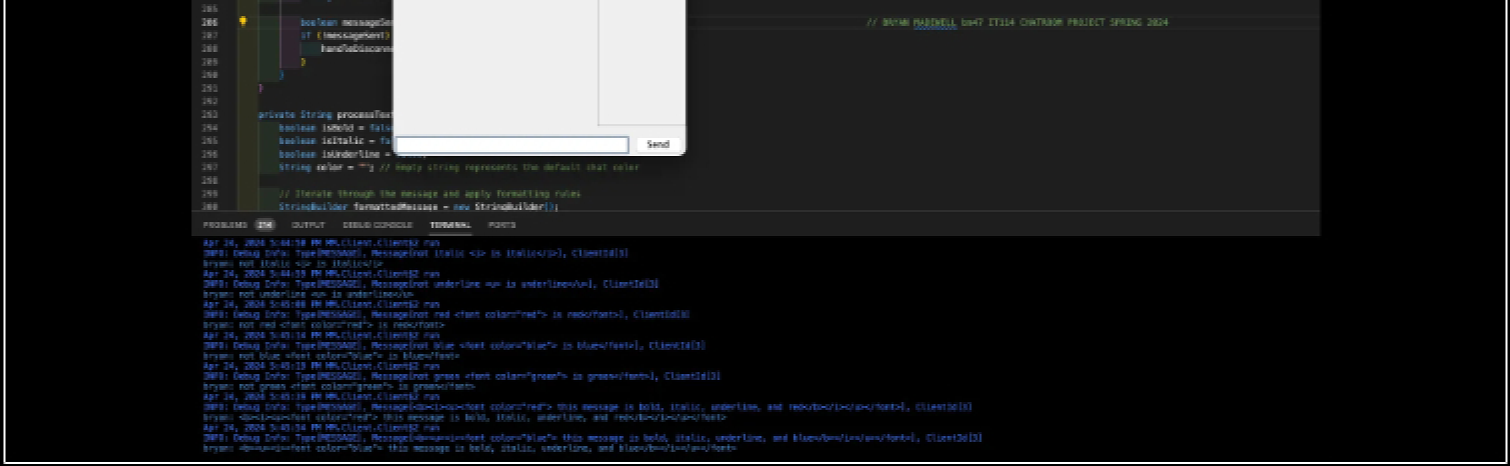
Task Screenshots:

Gallery Style: Large View

Small Medium Large



bryan connected
bryan: set bold **is bold**
bryan: set italic *is italic*
bryan: set underline <u>is underline</u>
bryan: set red is red
bryan: set blue is blue
bryan: set green is green
bryan: ***<u>this message is bold, italic, underline, and red</u>***
bryan: ***<u>this message is bold, italic, underline, and blue</u>***



```
            boolean messageSe                                                         // BRYAN MADEWELL bm47 IT114 CHATROOM PROJECT SPRING 2024
            if (!messageSent)
                handleDisconne
            }
        }
    }

    private String processText
        boolean isBold = false
        boolean isItalic = fa
        boolean isUnderline =
        String color = ""; // Empty string represents the default chat color

        // Iterate through the message and apply formatting rules
        StringBuilder formattedMessage = new StringBuilder();
```

PROBLEMS 250 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Apr 24, 2024 5:44:50 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[not italic <i> is italic</i>], ClientId[3]
bryan: not italic <i> is italic</i>
Apr 24, 2024 5:44:55 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[not underline <u> is underline</u>], ClientId[3]
bryan: not underline <u> is underline</u>
Apr 24, 2024 5:45:00 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[not red <font color="red"> is red</font>], ClientId[3]
bryan: not red <font color="red"> is red</font>
Apr 24, 2024 5:45:14 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[not blue <font color="blue"> is blue</font>], ClientId[3]
bryan: not blue <font color="blue"> is blue</font>
Apr 24, 2024 5:45:19 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[not green <font color="green"> is green</font>], ClientId[3]
bryan: not green <font color="green"> is green</font>
Apr 24, 2024 5:45:28 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[<b><i><u><font color="red"> this message is bold, italic, underline, and red</b></i></u></font>], ClientId[3]
bryan: <b><i><u><font color="red"> this message is bold, italic, underline, and red</b></i></u></font>
Apr 24, 2024 5:45:34 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[<b><u><i><font color="blue"> this message is bold, italic, underline, and blue</b></i></u></font>], ClientId[3]
bryan: <b><u><i><font color="blue"> this message is bold, italic, underline, and blue</b></i></u></font>
```

All of the features are included, and the message describe what they are. I showed the fact that some of the message can be a color, while the other not. I also included the terminal in the bottom for additional proof of the fully functioning text formatting, and the UCID on the right side of the VScode file.

Checklist Items (0)

●

^COLLAPSE ^

Task #3 - Points: 1

Text: Screenshot of the code solving the formatting display

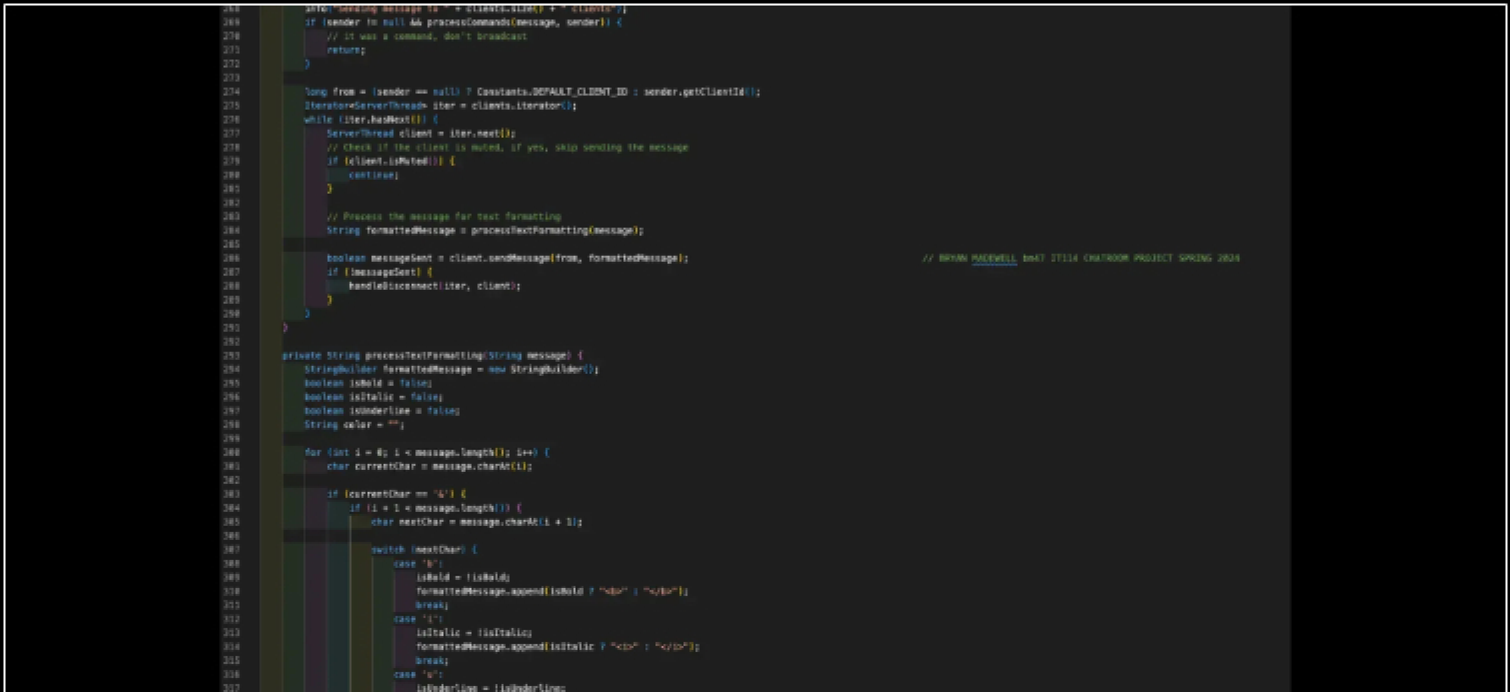
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show each relevant file this was done in (may be one or more)
#2	1	Include ucid and date comment
#3	1	Clearly caption screenshots

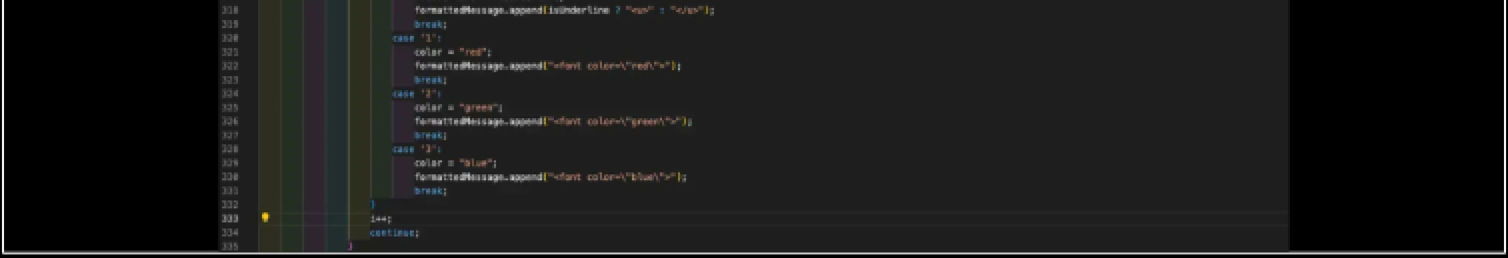
Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
        info("Sending message to " + clients.size() + " clients");
        if (sender != null && processCommands(message, sender)) {
            // it was a command, don't broadcast
            return;
        }

        long from = (sender == null) ? Constants.DEFAULT_CLIENT_ID : sender.getClientId();
        Iterator<ServerThread> iter = clients.iterator();
        while (iter.hasNext()) {
            ServerThread client = iter.next();
            // Check if the client is muted, if yes, skip sending the message
            if (client.isMuted()) {
                continue;
            }

            // Process the message for text formatting
            String formattedMessage = processTextFormatting(message);

            boolean messageSent = client.sendMessage(from, formattedMessage);                 // BRYAN MADEWELL bm47 IT114 CHATROOM PROJECT SPRING 2024
            if (!messageSent) {
                handleDisconnect(iter, client);
            }
        }
    }

    private String processTextFormatting(String message) {
        StringBuilder formattedMessage = new StringBuilder();
        boolean isBold = false;
        boolean isItalic = false;
        boolean isUnderline = false;
        String color = "";

        for (int i = 0; i < message.length(); i++) {
            char currentChar = message.charAt(i);

            if (currentChar == '%') {
                if (i + 1 < message.length()) {
                    char nextChar = message.charAt(i + 1);

                    switch (nextChar) {
                        case 'b':
                            isBold = !isBold;
                            formattedMessage.append(isBold ? "<b>" : "</b>");
                            break;
                        case 'i':
                            isItalic = !isItalic;
                            formattedMessage.append(isItalic ? "<i>" : "</i>");
                            break;
                        case 'u':
                            isUnderline = !isUnderline;
```



```
                    formattedMessage.append(isUnderline ? "<u>" : "</u>");
                    break;
                case '1':
                    color = "red";
                    formattedMessage.append("<font color=\"red\">");
                    break;
                case '2':
                    color = "green";
                    formattedMessage.append("<font color=\"green\">");
                    break;
                case '3':
                    color = "blue";
                    formattedMessage.append("<font color=\"blue\">");
                    break;
            }
            i++;
            continue;
        }
```

For this code in Room.java, I used switch statements to simplify the process of text formatting. I explain it in the question asking about the details.

Checklist Items (0)

● ^COLLAPSE^

Task #4 - Points: 1

Text: Explain how the formatting was made to be visible/rendered in the UI

Details:
Note each scenario

Response:

Most of this work is done in the Room.java class. As stated in class, the ChatPanel.java file should have been changed from plain to HTML text, which is one of the most important reasons this works. What I did to get it to work, was use the & symbol to dictate what formatting would be used. The way I pictured this working was the & was essentially the HTML tags, but looking much neater. For example, &b hey &b hey would be hey in bold, and then hey in regular text. I did this by creating a processTextFormatting method that checks every message sent, and depending on what comes after the &, the text formatting is then applied. For simplicity, I made bold B, italics I, and underline U. I also made red 1, green 2, and blue 3. This makes it easier to remember for me because of the color acronym RGB (red green blue).

● Private Message with @ (2 pts.)
^COLLAPSE^

● ^COLLAPSE^

Task #1 - Points: 1

Text: Screenshots demoing private message

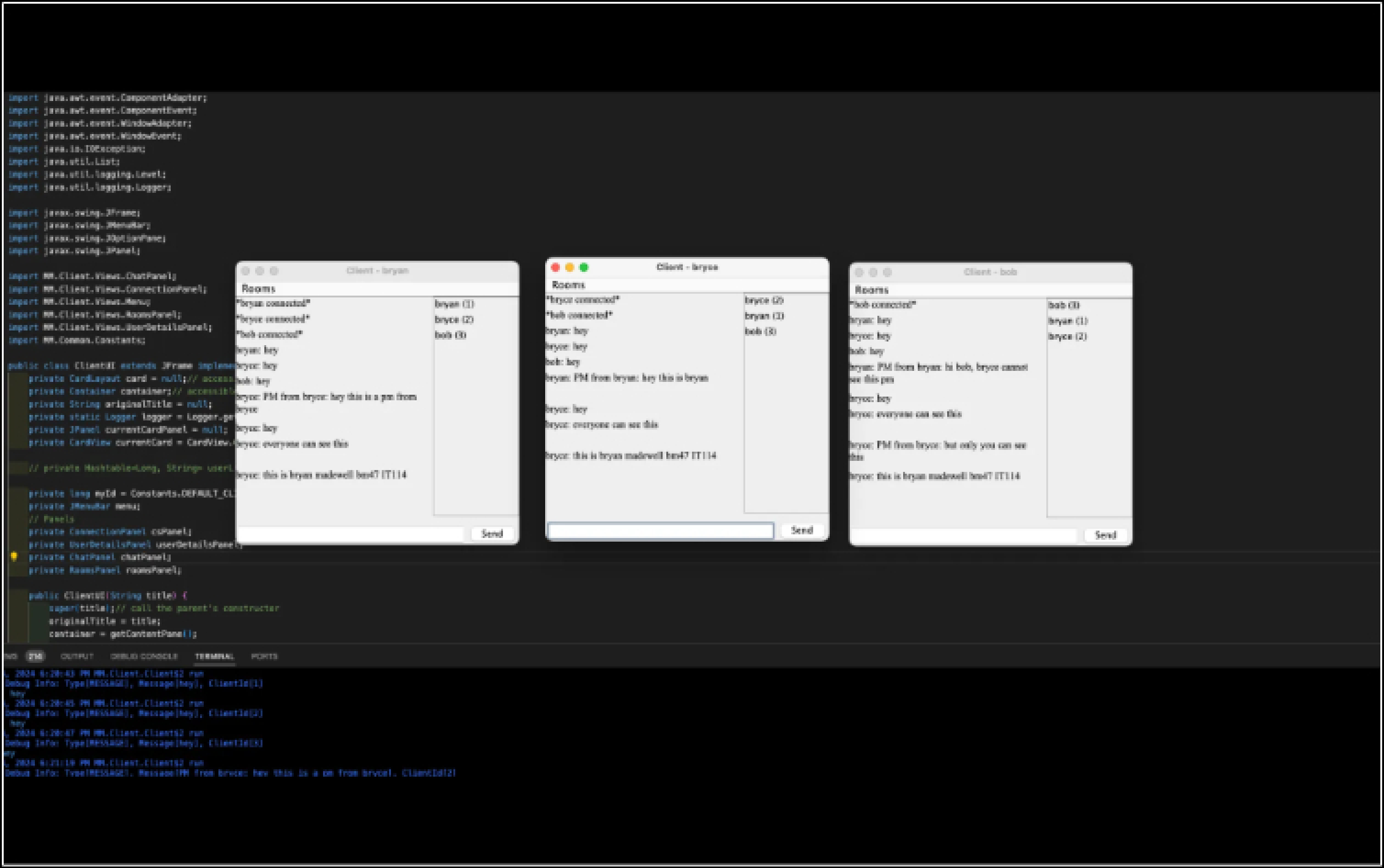
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Should have 3 clients in the same room
☐ #2	1	Demo a private message where only the sender and target see the message
☐ #3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.JFrame;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import MM.Client.Views.ChatPanel;
import MM.Client.Views.ConnectionPanel;
import MM.Client.Views.Menu;
import MM.Client.Views.RoomsPanel;
import MM.Client.Views.UserDetailsPanel;
import MM.Common.Constants;

public class ClientUI extends JFrame implemen
    private CardLayout card = null;// access
    private Container container;// accessib
    private String originalTitle = null;
    private static Logger logger = Logger.ge
    private JPanel currentCardPanel = null;
    private CardView currentCard = CardView.

    // private Hashtable<Long, String> userL

    private long myId = Constants.DEFAULT_CL
    private JMenuBar menu;
    // Panels
    private ConnectionPanel csPanel;
    private UserDetailsPanel userDetailsPanel;
    private ChatPanel chatPanel;
    private RoomsPanel roomsPanel;

    public ClientUI(String title) {
        super(title);// call the parent's constructor
        originalTitle = title;
        container = getContentPane();
```

Client - bryan

Rooms

```
*bryan connected*                           bryan (1)
*bryce connected*                           bryce (2)
*bob connected*                             bob (3)
bryan: hey
bryce: hey
bob: hey
bryce: PM from bryce: hey this is a pm from
bryce
bryce: hey
bryce: everyone can see this

bryce: this is bryan madewell bm47 IT114
```

Client - bryce

Rooms

```
*bryce connected*                           bryce (2)
*bob connected*                             bryan (1)
bryan: hey                                  bob (3)
bryce: hey
bob: hey
bryan: PM from bryan: hey this is bryan

bryce: hey
bryce: everyone can see this

bryce: this is bryan madewell bm47 IT114
```

Client - bob

Rooms

```
*bob connected*                             bob (3)
bryan: hey                                  bryan (1)
bryce: hey                                  bryce (2)
bob: hey
bryan: PM from bryan: hi bob, bryce cannot
see this pm
bryce: hey
bryce: everyone can see this

bryce: PM from bryce: but only you can see
this
bryce: this is bryan madewell bm47 IT114
```

```
, 2024 6:20:43 PM MM.Client.Client$2 run
Debug Info: Type[MESSAGE], Message[hey], ClientId[1]
 hey
, 2024 6:20:45 PM MM.Client.Client$2 run
Debug Info: Type[MESSAGE], Message[hey], ClientId[2]
 hey
, 2024 6:20:47 PM MM.Client.Client$2 run
Debug Info: Type[MESSAGE], Message[hey], ClientId[3]
hey
, 2024 6:21:10 PM MM.Client.Client$2 run
Debug Info: Type[MESSAGE], Message[PM from bryce: hey this is a pm from bryce], ClientId[2]
```

All requirements met

Checklist Items (0)

^COLLAPSE ^

Task #2 - Points: 1

Text: Screenshots of the related code

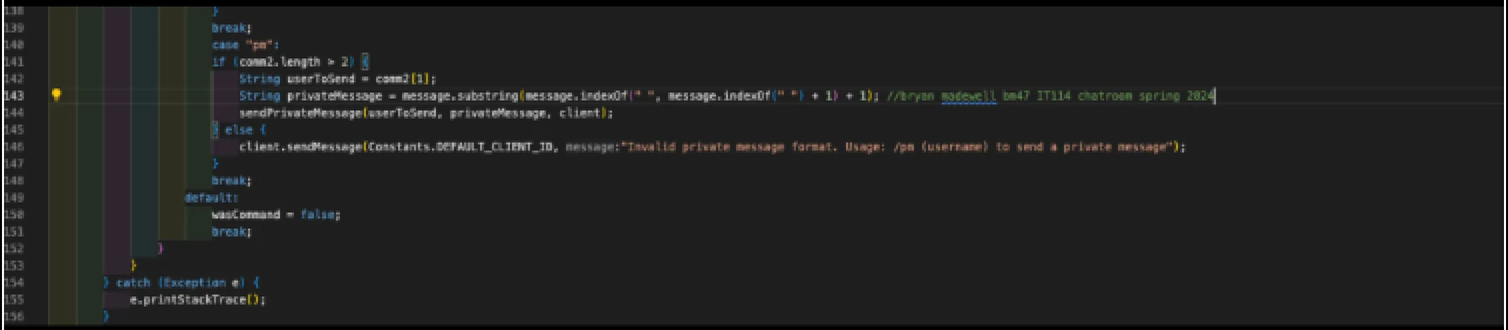
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show what code processes and handles the private message
#2	1	The message should only be sent to the receiver and the target
#3	1	The client should be targeting the username and the server side should be fetching the correct recipient
#4	1	Include ucid and date comment
#5	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
                }
                break;
                case "pm":
                if (comm2.length > 2) {
                    String userToSend = comm2[1];
                    String privateMessage = message.substring(message.indexOf(" ", message.indexOf(" ") + 1) + 1); //bryan madewell bm47 IT114 chatroom spring 2024
                    sendPrivateMessage(userToSend, privateMessage, client);
                } else {
                    client.sendMessage(Constants.DEFAULT_CLIENT_ID, message:"Invalid private message format. Usage: /pm (username) to send a private message");
                }
                break;
            default:
                wasCommand = false;
                break;
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

This is the most relevant code when it comes to the private messages. I had a difficult time getting @ to work, so I decided to do another command, that being /pm, hence the "pm" case. UCID and date included and the proof of this working is in the other screenshot where it asks to show that it works.

Checklist Items (0)

●

^COLLAPSE^

Task #3 - Points: 1

Text: Explain how private message works related to the code above

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Include how the sender and receiver are handled
#2	1	Include how the username is used to get the proper id

Response:

The command /pm triggers the private message method, which then first checks and sees if there is a valid user in the room intended to receive the private message. If the user exists, the message is then ONLY send to the user that it is intended for. When a client joins the room, an ID is given. Doing /pm (name) or @(name) will be using the clientName, but the clientName is also related to the clientID, that it how the clientID is retrieved from this.

●

Mute/Unmute Users (3 pts.)

^COLLAPSE^

●

^COLLAPSE^

Task #1 - Points: 1

Text: Screenshots demoing feature working

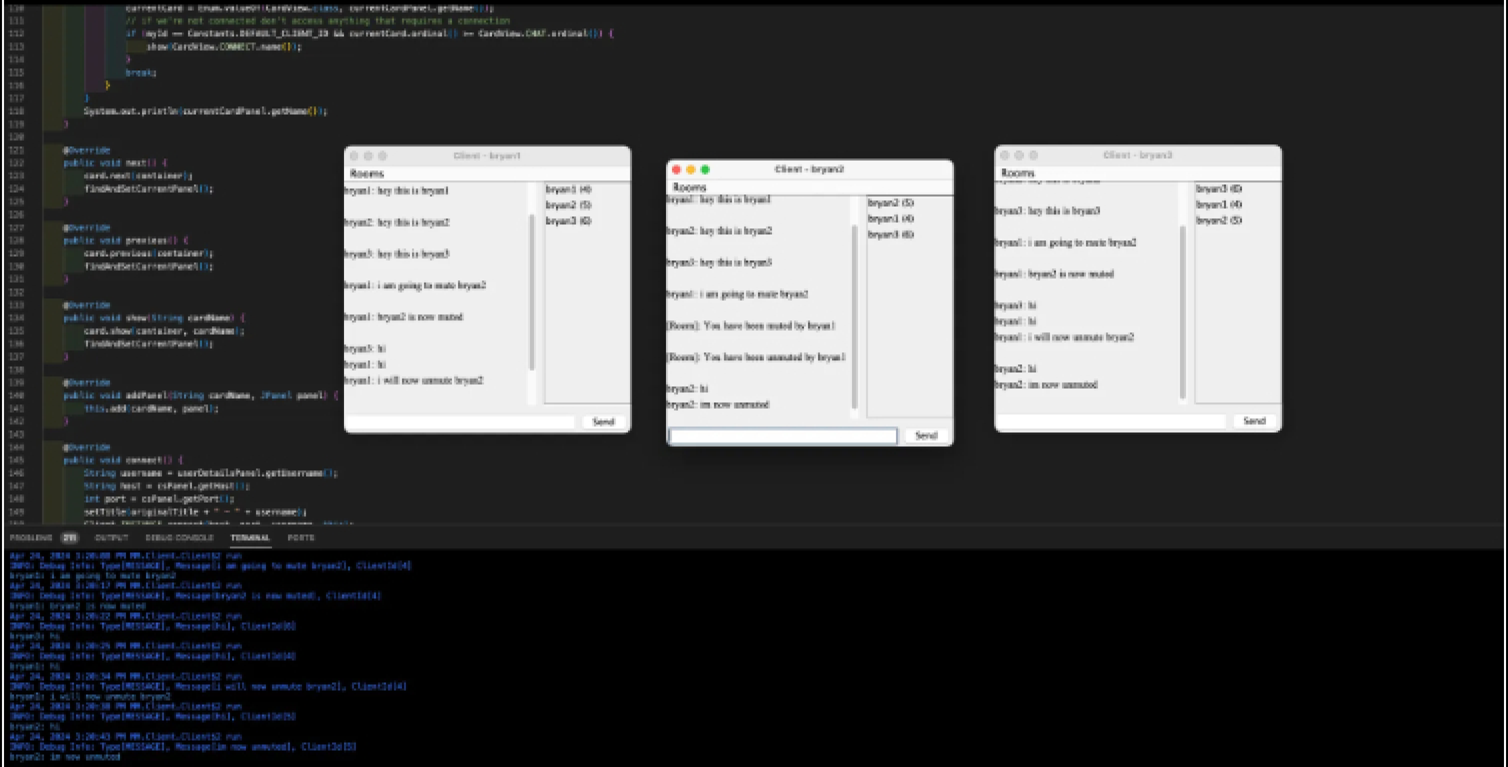
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have 3 clients in the same room
#2	1	Demo mute preventing messages between the muter and the target
#3	1	Demo mute also being accounted for with private messages
#4	1	Demo unmute allowing the messages again from the target to the unmuter

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
                currentCard = Enum.valueOf(CardView.class, currentCardPanel.getName());
                // if we're not connected don't access anything that requires a connection
                if (myId == Constants.DEFAULT_CLIENT_ID && currentCard.ordinal() >= CardView.CHAT.ordinal()) {
                    show(CardView.CONNECT.name());
                }
                break;
            }
        }
        System.out.println(currentCardPanel.getName());
    }

    @Override
    public void next() {
        card.next(container);
        findAndSetCurrentPanel();
    }

    @Override
    public void previous() {
        card.previous(container);
        findAndSetCurrentPanel();
    }

    @Override
    public void show(String cardName) {
        card.show(container, cardName);
        findAndSetCurrentPanel();
    }

    @Override
    public void addPanel(String cardName, JPanel panel) {
        this.add(cardName, panel);
    }

    @Override
    public void connect() {
        String username = userDetailsPanel.getUsername();
        String host = ciPanel.getHost();
        int port = ciPanel.getPort();
        setTitle(originalTitle + " - " + username);
```

Client - bryan1

Rooms

```
bryan1 : hey this is bryan1

bryan2: hey this is bryan2

bryan3: hey this is bryan3

bryan1 : i am going to mute bryan2

bryan1 : bryan2 is now muted

bryan3: hi
bryan1 : hi
bryan1 : i will now unmute bryan2
```

bryan1 (4)
bryan2 (5)
bryan3 (6)

Send

Client - bryan2

Rooms

```
bryan1: hey this is bryan1

bryan2: hey this is bryan2

bryan3: hey this is bryan3

bryan1 : i am going to mute bryan2

[Room]: You have been muted by bryan1

[Room]: You have been unmuted by bryan1

bryan2: hi
bryan2: im now unmuted
```

bryan2 (5)
bryan1 (4)
bryan3 (6)

Send

Client - bryan3

Rooms

```
bryan3 : hey this is bryan3

bryan1 : i am going to mute bryan2

bryan1 : bryan2 is now muted

bryan3 : hi
bryan1 : hi
bryan1 : i will now unmute bryan2

bryan2 : hi
bryan2 : im now unmuted
```

bryan3 (6)
bryan1 (4)
bryan2 (5)

Send

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Apr 24, 2024 3:20:08 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[i am going to mute bryan2], ClientId[4]
bryan1: i am going to mute bryan2
Apr 24, 2024 3:20:17 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[bryan2 is now muted], ClientId[4]
bryan1: bryan2 is now muted
Apr 24, 2024 3:20:22 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[hi], ClientId[6]
bryan3: hi
Apr 24, 2024 3:20:25 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[hi], ClientId[4]
bryan1: hi
Apr 24, 2024 3:20:34 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[i will now unmute bryan2], ClientId[4]
bryan1: i will now unmute bryan2
Apr 24, 2024 3:20:38 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[hi], ClientId[5]
bryan2: hi
Apr 24, 2024 3:20:43 PM MM.Client.Client$2 run
INFO: Debug Info: Type[MESSAGE], Message[im now unmuted], ClientId[5]
bryan2: im now unmuted
```

This screenshot shows 3 users in the room, all with the ability to speak. Bryan1 then mutes Bryan2 and the message is then sent to the muted user letting them know they have been muted. Bryan1 then UNmutes Bryan2 and it is shown in the chat he is unmuted and Bryan2 then can send a message again.

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the related code

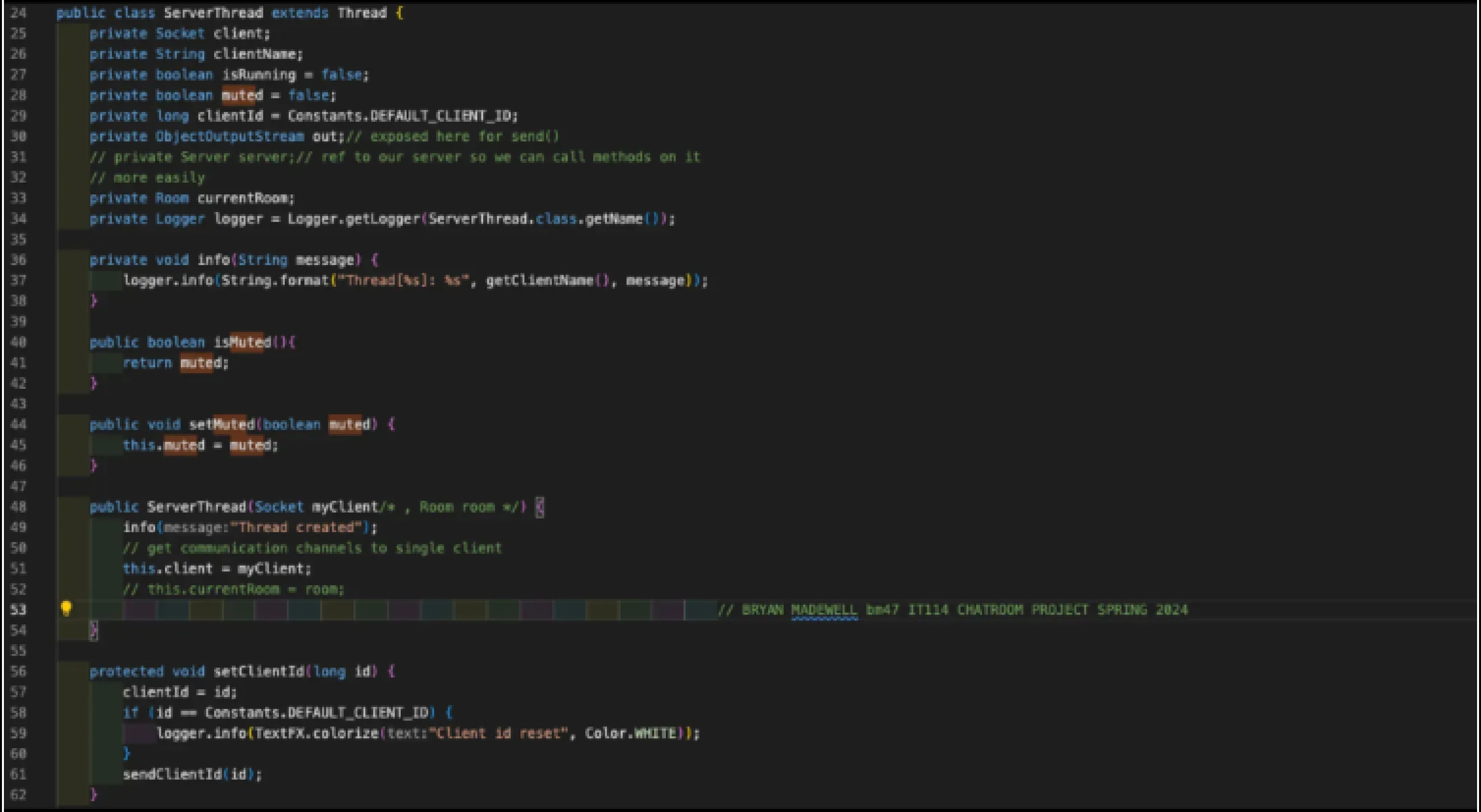
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	ServerThread should have a list of who they muted
#2	1	ServerThread should expose and add, remove, and is muted check to room
#3	1	Room should handle the mute list when receiving the appropriate payloads
#4	1	Room should check the mute list during send message and private messages
#5	1	Include ucid and date comment
#6	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

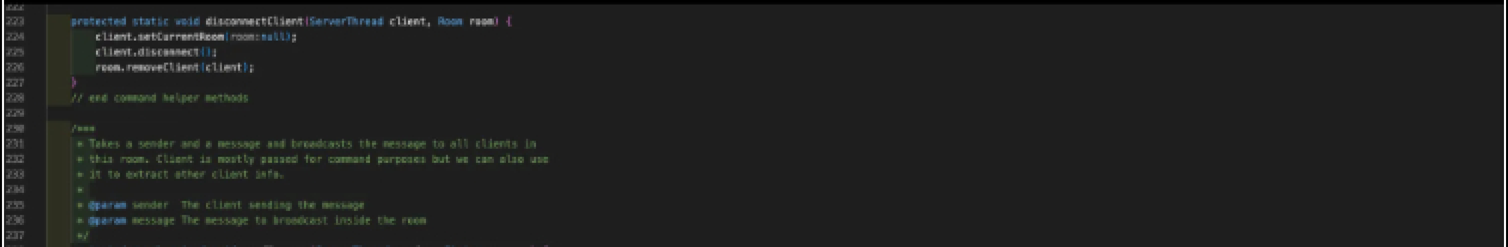
Small Medium Large



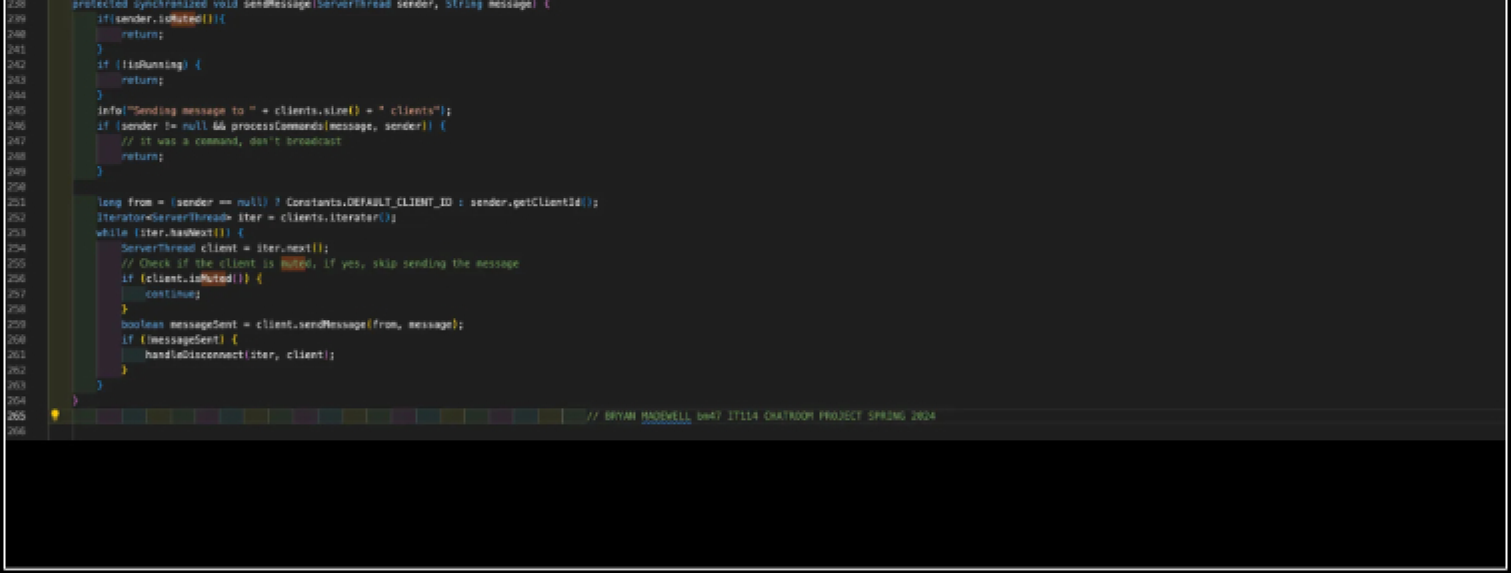
```java
public class ServerThread extends Thread {
    private Socket client;
    private String clientName;
    private boolean isRunning = false;
    private boolean muted = false;
    private long clientId = Constants.DEFAULT_CLIENT_ID;
    private ObjectOutputStream out;// exposed here for send()
    // private Server server;// ref to our server so we can call methods on it
    // more easily
    private Room currentRoom;
    private Logger logger = Logger.getLogger(ServerThread.class.getName());

    private void info(String message) {
        logger.info(String.format("Thread[%s]: %s", getClientName(), message));
    }

    public boolean isMuted(){
        return muted;
    }

    public void setMuted(boolean muted) {
        this.muted = muted;
    }

    public ServerThread(Socket myClient/* , Room room */) {
        info(message:"Thread created");
        // get communication channels to single client
        this.client = myClient;
        // this.currentRoom = room;
                                                    // BRYAN MADEWELL bm47 IT114 CHATROOM PROJECT SPRING 2024
    }

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

This screenshot shows some of the code used to mute a user. It uses isMuted() and setMuted() to check if a user is muted, and then to set a user as muted.

Checklist Items (0)



```java
    protected static void disconnectClient(ServerThread client, Room room) {
        client.setCurrentRoom(room:null);
        client.disconnect();
        room.removeClient(client);
    }
    // end command helper methods

    /**
     * Takes a sender and a message and broadcasts the message to all clients in
     * this room. Client is mostly passed for command purposes but we can also use
     * it to extract other client info.
     *
     * @param sender  The client sending the message
     * @param message The message to broadcast inside the room
     */
```



```java
    protected synchronized void sendMessage(ServerThread sender, String message) {
        if(sender.isMuted()){
            return;
        }
        if (!isRunning) {
            return;
        }
        info("Sending message to " + clients.size() + " clients");
        if (sender != null && processCommands(message, sender)) {
            // it was a command, don't broadcast
            return;
        }

        long from = (sender == null) ? Constants.DEFAULT_CLIENT_ID : sender.getClientId();
        Iterator<ServerThread> iter = clients.iterator();
        while (iter.hasNext()) {
            ServerThread client = iter.next();
            // Check if the client is muted, if yes, skip sending the message
            if (client.isMuted()) {
                continue;
            }
            boolean messageSent = client.sendMessage(from, message);
            if (!messageSent) {
                handleDisconnect(iter, client);
            }
        }
    }
                                                                    // BRYAN MADEWELL bm47 IT114 CHATROOM PROJECT SPRING 2024
```

Here is the Room.java code, the first screenshot being the ServerThread.java code. This shows more of the code used to mute the user, using isMuted to determine whether or not a user it muted. If the user is muted, the message is not sent to other users. If the user is not, they are allows to send a message to the room/lobby.

Checklist Items (0)

●

^COLLAPSE^

Task #3 - Points: 1

Text: Explain how the mute and unmute logic works in relation to the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Explain how your mute list is handled
☐ #2	1	Explain how it's handled/processed in send message and private message

Response:

Mute is handled in both the Room.java file and the ServerThread.java file. There are two methods mainly used, those being isMuted, and setMute. One the command /mute (user) is sent, it is checked to ensure that the user in the room exists. If the user does not exist, the command does not work and the user who executed the command is notified that the user does not exist. If it does work, it works via the setMute() method and then deems the user unable to send messages for the time being. Until the user is set to unmuted, they are not allowed to send messages.

●

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
Missing URL

● COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

Missing Response

● 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

Missing Caption

End of Assignment