Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-milestone-2-chatroom-2024-m24/grade/jah89

IT114-450-M2024 - [IT114] Milestone 2 Chatroom 2024 (M24)

Submissions:

Submission Selection

1 Submission [active] 7/3/2024 1:09:16 PM

Instructions

COLLAPSE

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

Branch name: Milestone2

Tasks: 8 Points: 10.00

Payloads (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Base Payload Class

Details:
All code screenshots must have ucid/date visible.

#1) Show screenshot of the Payload.java

```java
package Project.common;

import java.io.Serializable;

public class Payload implements Serializable {   //jub80 07 07 2024
    private PayloadType payloadType;
    private long clientId;
    private String message;

    public PayloadType getPayloadType() {
        return payloadType;
    }

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

    public long getClientId() {
        return clientId;
    }

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

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    @Override
    public String toString(){
        return String.format("Payload[%s] Client Id [%s] Message: [%s]", getPayloadType(), getClientId(), getMessage());
    }
}
```

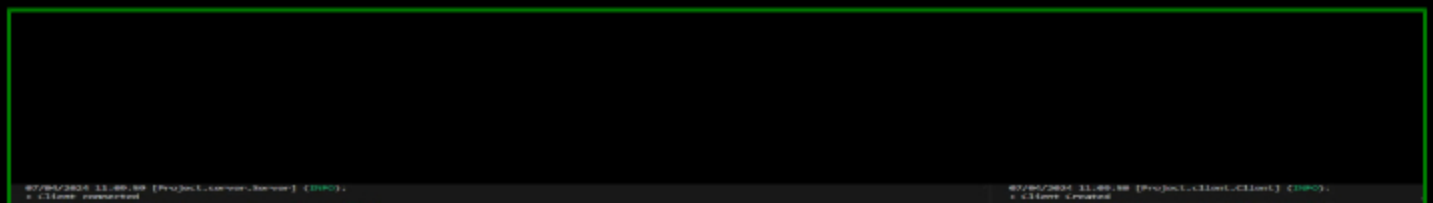
Caption (required) ✓
Describe/highlight what's being shown
Showing payload.java

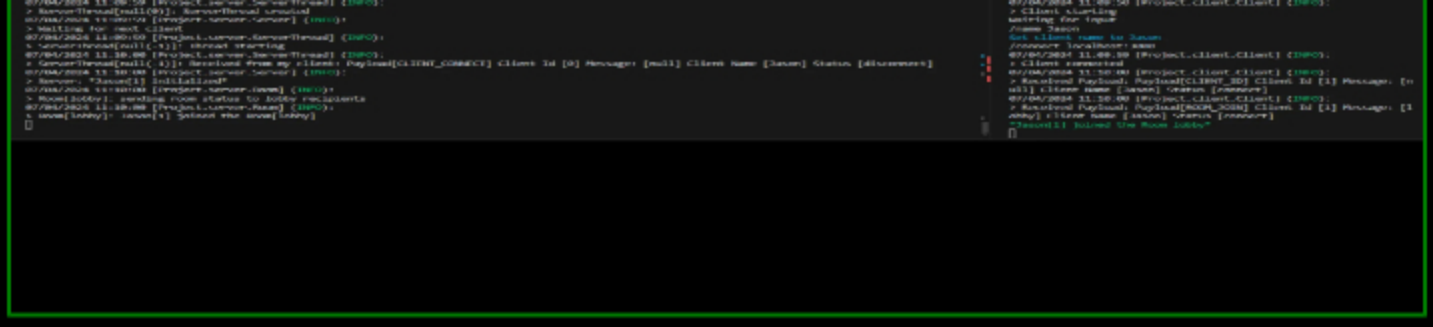
Explanation (required) ✓
Briefly explain the purpose of each property and serialization

PREVIEW RESPONSE

The purpose of the first payload (private PayloadType payloadType) is to store the type of payload that is being sent or recieved. This helps tell apart different types of messages or actions/commands. The next line (public long getClientID) is to store the unique ID of the client that is sending or recieving the payload. This is used to track which client is associated with the payload. The next line that is public String getMessage, has the purpose of holding the actual message or data being sent in the payload. Depending on the payload type it can hold different types of information. When we implement Serializable it allows the payload class to be serialized. The purpose of serialization is to convert an object into a byte stream, which can be easily transmitted over a network. It allows for the payload objects to be converted to a format that can be easily sent and reconstructed on the other side.

#2) Show screenshot examples of the terminal output for base Payload objects





Caption (required) ✓

Describe/highlight what's being shown

Showing terminal showing base payloads.

COLLAPSE

Task #2 - Points: 1

Text: RollPayload Class

ℹ Details:

All code screenshots must have ucid/date visible.

#1) Show screenshot of the RollPayload.java (or equivalent)

```java
package Project.common;
//jah80 07/03/2024
public class RollPayload extends Payload {
    private int sides;
    private int rolls;

    public int getSides() {
        return sides;
    }

    public void setSides(int sides) {
        this.sides = sides;
    }
    public int getRolls() {
        return rolls;
    }

    public void setRolls(int rolls) {
        this.rolls = rolls;
    }

    @Override
    public String toString() {
        return String.format("RollPayload[sides=%d, rolls=%d]", sides, rolls);
    }
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing RollPayload class

Explanation (required) ✓

Briefly explain the purpose of each property

PREVIEW RESPONSE

I used private int sides to store the number of sides on the dice before being rolled which would store 6 for a dice. I

then used private int rolls to store the number of dice being rolled, so if two dice are being rolled it would be set to 2. I used public int getSides() to return the value of the 'sides' to show many many sides the dice has. And then I use setSides to set the value to sides. I use getRolls() to return the value of how many dice are being rolled and then use setRolls(int rolls) to set the value. Finally I added an override toString method to represent the RollPayload object which holds the number of sides and number of rolls.

#2) Show screenshot examples of the terminal output for base RollPayload objects

Caption (required) ✓

Describe/highlight what's being shown

Showing terminal while using /roll

Client Commands (4 pts.)

^COLLAPSE^

^COLLAPSE^

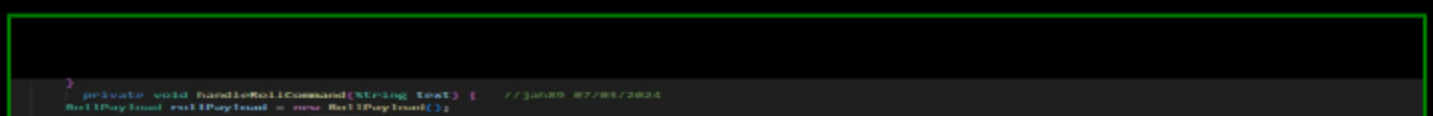
Task #1 - Points: 1

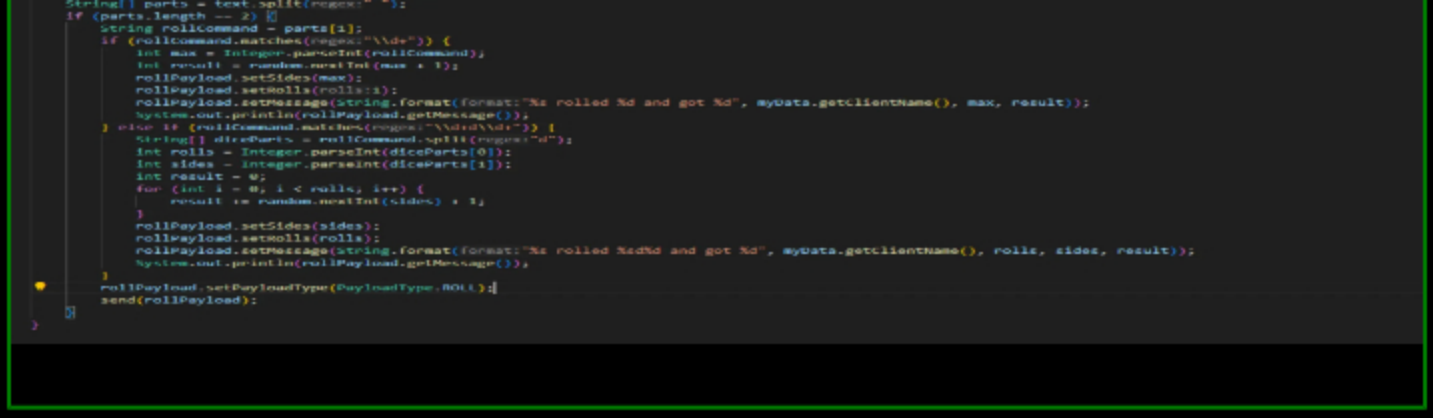
Text: Roll Command

Details:

All code screenshots must have ucid/date visible.
Any output screenshots must have at least 3 connected clients able to see the output.
All commands must show who triggered it, what they did (specifically) and what the outcome was.

#1) Show the client side code for handling /roll #





Caption (required) ✓

Describe/highlight what's being shown

Showing code to handle roll command

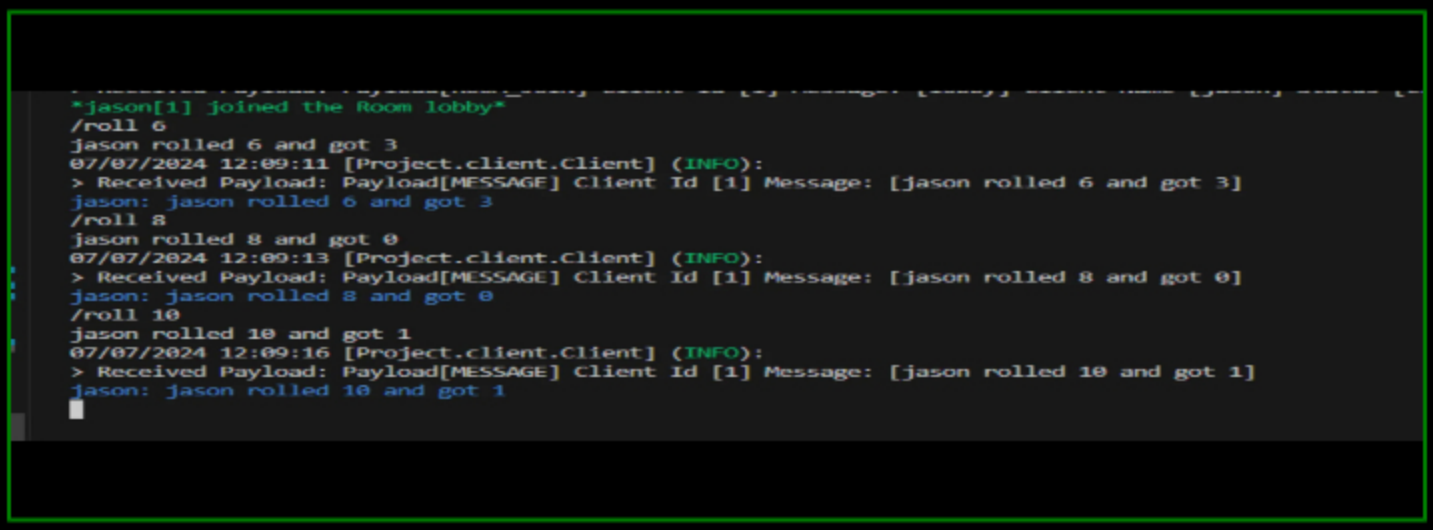
Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

In the first line within the method (RollPayload rollPayload = new RollPayload();) I created a new RollPayload object to store the details of the roll. I then split the text to extract the command and the arguments that were typed. I then added an if statement to check the command is in the correct format by using if(parts.length == 2)....; This makes sure the command has 2 parts which is the command itself such as /roll and then the second argument either a number(/roll 10) or 2d6. If the argument given is a single number it will parse it as the maximum value. It will then generate a random result between 0 and 'max'. It then sets the roll details in "rollPayload" and prints the result. It then uses rollPayload.setPayloadType(PayloadType.ROLL); to set the payload type and send the payload using send(rollPayload).

#2) Show the output of a few examples of /roll # (related payload output should be visible)



Caption (required) ✓

Describe/highlight what's being shown

Showing /roll # examples

#3) Show the client side code for handling /roll #d# (related payload output should be visible)

```
        System.out.println(rollPayload.getMessage());
    } else if (rollCommand.matches(regex:"\\d+d\\d+")) {  //Jah09 07-07-2024
        String[] diceParts = rollCommand.split(regex:"d");
        int rolls = Integer.parseInt(diceParts[0]);
        int sides = Integer.parseInt(diceParts[1]);
        int result = 0;
        for (int i = 0; i < rolls; i++) {
            result += random.nextInt(sides) + 1;
        }
        rollPayload.setSides(sides);
        rollPayload.setRolls(rolls);
        rollPayload.setMessage(String.format(format:"%s rolled %sd%d and got %d", myData.getClientName(), rolls, sides, result));
        System.out.println(rollPayload.getMessage());
    }
    rollPayload.setPayloadType(PayloadType.ROLL);
    send(rollPayload);
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code for /roll #d#

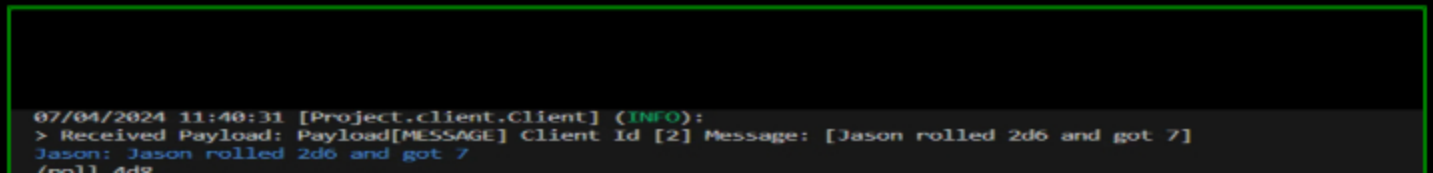
Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

In this code the first thing I do is match the command by doing "else if (rollCommand.matches("\d+d\d+"))". This will check if the command matches the #d# format where # represents a digit sent by the client. I then parse through the command by using parseInt which will split the command into two parts at the 'd' character. The number of rolls and number of sides are also parsed from the command. I then ran a loop for the number of rolls, in each replay of the loop a random number between 1 and # of sides given is generated and added to the result. I then set the roll details in Rollpayload by doing rollPayload.setSides(sides); I did this for setRolls as well. I then formatted the message using %s to represent the client that committed the command and then %sd%d where %d holds the integer of number of sides and number of rolls. I gave it the argument "myData.getClientName(), rolls, sides, result" to replace the placeholders in the format string. Finally in the last snippet of code I print and send the payload. I print the roll result and set the payload to "ROLL" and then send rollPayload to the server usin send(rollPayload).

#4) Show the output of a few examples of /roll #d#



```
07/04/2024 11:40:31 [Project.client.Client] (INFO):
> Received Payload: Payload[MESSAGE] Client Id [2] Message: [Jason rolled 2d6 and got 7]
Jason: Jason rolled 2d6 and got 7
(poll 4d8
```

```
Jason rolled 4d8 and got 19
07/04/2024 11:40:37 [Project.client.Client] (INFO):
> Received Payload: Payload[MESSAGE] Client Id [2] Message: [Jason rolled 4d8 and got 19]
Jason: Jason rolled 4d8 and got 19
/roll 1d6
Jason rolled 1d6 and got 4
07/04/2024 11:40:43 [Project.client.Client] (INFO):
> Received Payload: Payload[MESSAGE] Client Id [2] Message: [Jason rolled 1d6 and got 4]
Jason: Jason rolled 1d6 and got 4
```

Caption (required) ✓

Describe/highlight what's being shown

Showing output of using /roll #d#

#5) Show the ServerThread code receiving the RollPayload

```
            RollPayload rollPayload = (RollPayload) payload;     //jah89 07/04/2024
            currentRoom.processRollCommand(this, rollPayload);
            break;
        case FLIP:
            currentRoom.processFlipCommand(this, payload);
            break;
        default:
            break;
    }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code in serverthread recieving payload

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The first line I am using 'payload' to cast a RollPayload object. The next line is the processRollCommand method so the current room is called and passes the current serverthread and rollpayload as arguments. I then added to the previous cases but added a case for the ROLL to specifically handle RollPayload objects.

#6) Show the Room code that processes both Rolls and sends the response

```
    //jah89 07-04-2024
protected synchronized void processRollCommand(ServerThread sender, RollPayload rollPayload) {
        String message = rollPayload.getMessage();
        sendMessage(sender, message);
    }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code from room.java that processes and sends response

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The first of line of code makes a processRollCommand method that uses ServerThread and Sender as arguments which represents the client that sent the roll command. The next argument RollPayload rollPayload is to hold the details of the roll comand, such as the number of sides or how many time it was rolled. The next line String Message is used to extract the message from rollpayload which will contain the result of the roll formatted as a string. The last line sendMessage sends the message to all clients in the room. The two parameters it takes is sender and message. Sender is the client who used the command and message is the message that will be sent out notifying the clients in the room the result of the dice roll.

^COLLAPSE ^

Task #2 - Points: 1

Text: Flip Command

#1) Show the client side code for handling /flip

```
    }
    //jah89 07-04-2024
    private void handleFlipCommand() {
        Payload flipPayload = new Payload();
        flipPayload.setPayloadType(PayloadType.FLIP);
        String result = random.nextBoolean() ? "heads" : "tails";
        flipPayload.setMessage(String.format(format:"%s flipped a coin and got %s", myData.getClientName(), result));
        System.out.println(flipPayload.getMessage());
        send(flipPayload);
        // end payload processors

    }
    }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing client side code to handle /flip

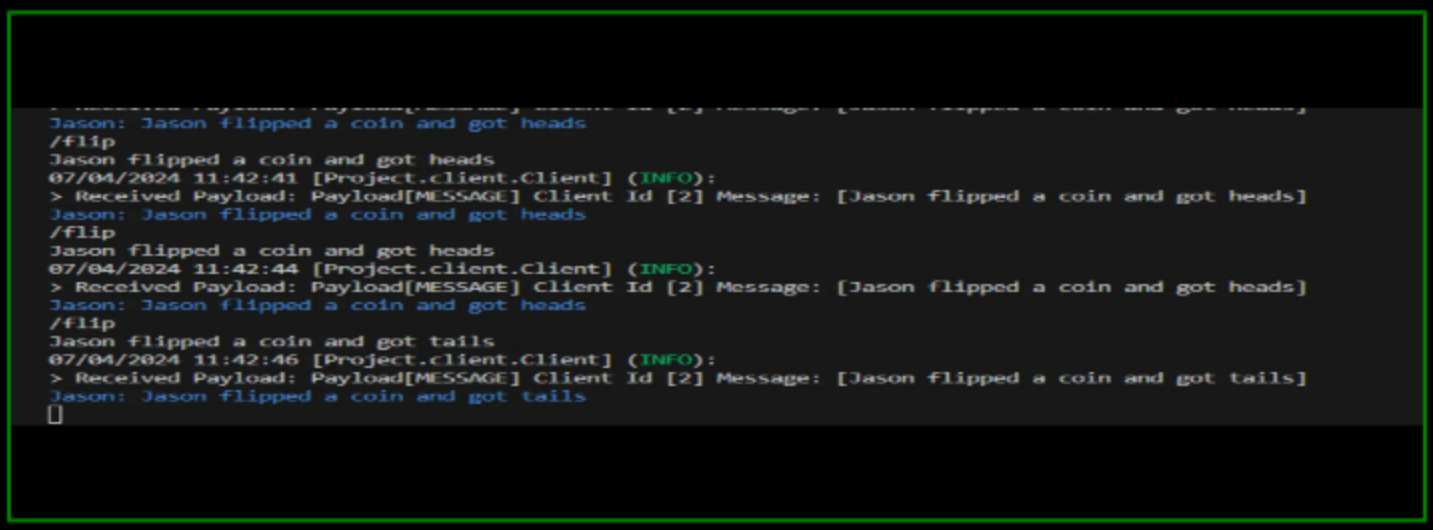
Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

Firstly I created a method called handleFlipCommand with a void because it does not return a value. The first step in this method was to create a new payload by using Payload flipPayload = new Payload(). This will create a new instance of the 'Payload' class to hold the information of the flip command. The next line I wanted to set the payload type as FLIP to show this payload is for the flip command. In the third line I have a random boolean generator which instead of true or false it is heads for true or tails for false. In the fourth line I set the message in the payload, this message includes the clients name and the result of the coin flip. I used string.format to format the message string with clients name and flip result. I then wanted to print the message to the console by using system.out.print(flipPayload.getMessage()); Finally I want to send the payload using send(flipPayload) which will send it to the server to be sent to all other clients.

#2) Show the output of a few examples of /flip (related payload output should be visible)



Caption (required) ✓

Describe/highlight what's being shown

Showing /flip command being used

Text Formatting (3 pts.)

COLLAPSE

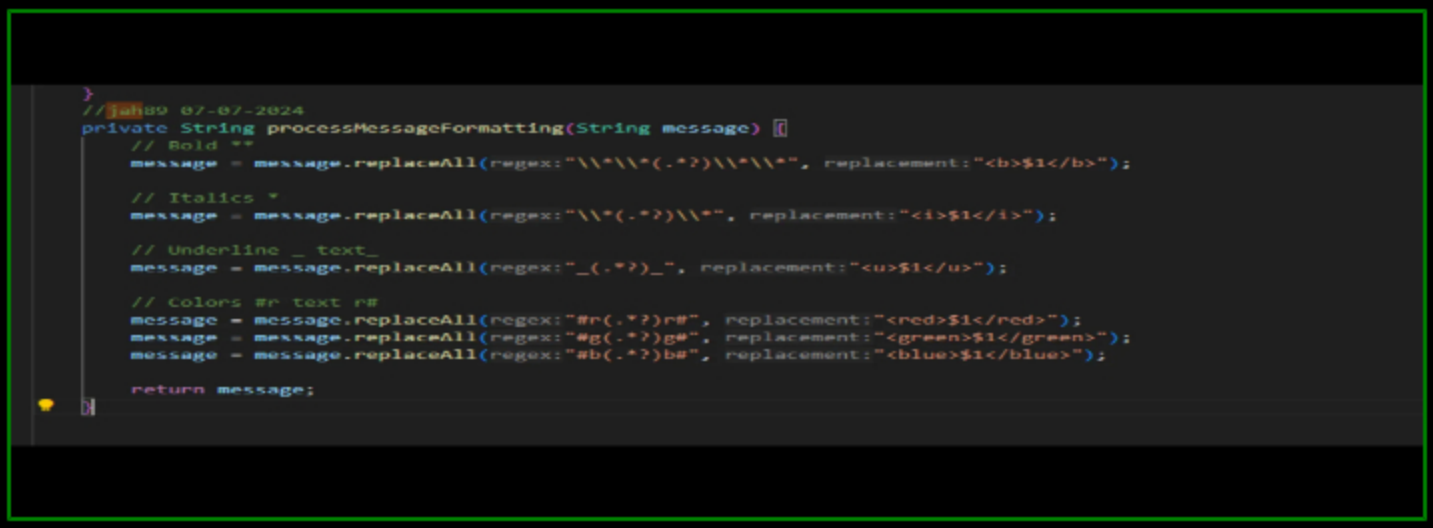
COLLAPSE

Task #1 - Points: 1

Text: Text Formatting

Details:
All code screenshots must have ucid/date visible.
Any output screenshots must have at least 3 connected clients able to see the output.
Note: Having the user type out html tags is not valid for this feature, instead treat it like WhatsApp, Discord, Markdown, etc
Note: Each text trigger must wrap the text that you want to affect
Note: Slash commands are not an accepted solution, the text must be transformed
Note: You do not need to use the same symbols in the below example, it's just an example, also, the below example doesn't show the "correct" output for colors, I'm leaving the proper conversion up to research on your own.
See proposal for an example.

#1) Show the code related to processing the special characters for bold, italic, underline, and colors, and converting them to other characters (should be in Room.java)

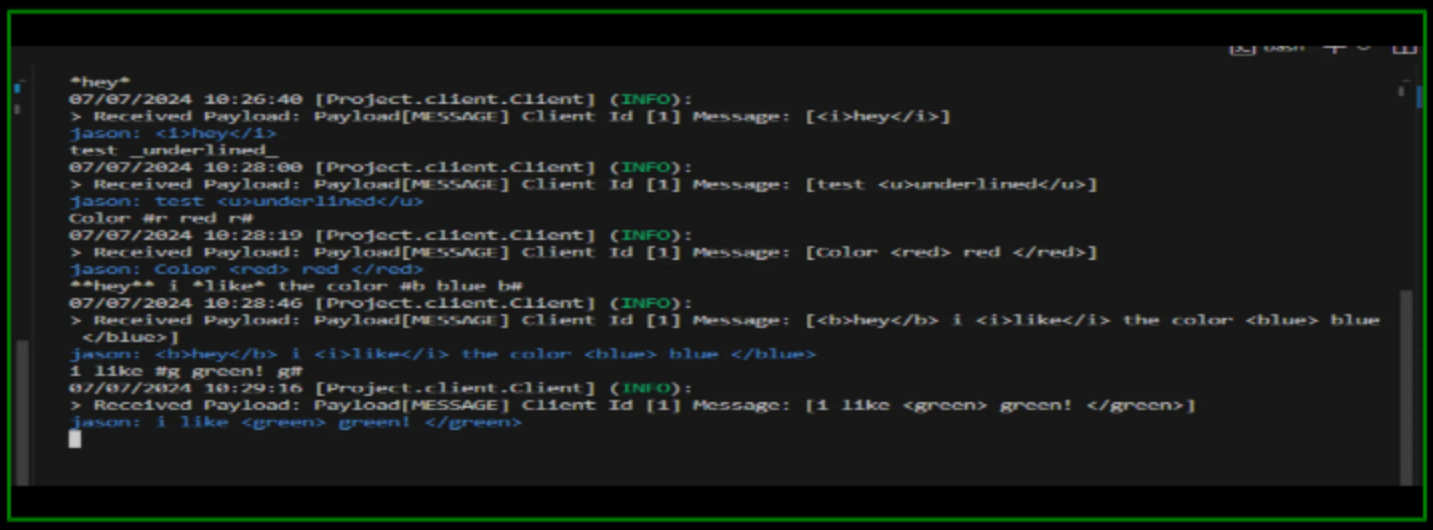


```
    }
    //jah89 07-07-2024
    private String processMessageFormatting(String message) {
        // Bold **
        message = message.replaceAll(regex:"\\*\\*(.*?)\\*\\*", replacement:"<b>$1</b>");

        // Italics *
        message = message.replaceAll(regex:"\\*(.*?)\\*", replacement:"<i>$1</i>");

        // Underline _ text_
        message = message.replaceAll(regex:"_(.*?)_", replacement:"<u>$1</u>");

        // Colors #r text r#
        message = message.replaceAll(regex:"#r(.*?)r#", replacement:"<red>$1</red>");
        message = message.replaceAll(regex:"#g(.*?)g#", replacement:"<green>$1</green>");
        message = message.replaceAll(regex:"#b(.*?)b#", replacement:"<blue>$1</blue>");

        return message;
    }
```

Caption (required) ✓
Describe/highlight what's being shown
Showing code added for special characters

Explanation (required) ✓
Briefly explain how it works and the choices of the placeholder characters and the result characters
PREVIEW RESPONSE
For the first line on bold formatting I made it so when a message is typed between 2 asteriks it will have the bold formatting. By using regex and replacement "\\(.?)\\", "**$1**"); This line of code is an expression that matches any text within the double asteriks and gives it replacement within **$1** making the message bold. In the regex the "* * " is the literal character . Then I used (. * ?) to capture the message inbetween the asteriks. And finally in the replacement I used < b > $1 which **represents opening bold tag and then $1 refers to the text captured in the regex. This same concept is applied to italics and underline except that the symbols used to identify each are changed. For example underlined characters start and end with an underscore and italics just use one asterik instead of two.**
For the colors in the regex I used "#r(*.?)r#" to represent each color, for examle #r is red and #b is blue etc. With the same conceptas above i used (.?) to capture the message typed in between these tags. It then replaces the message with the tagline and $1 to represent the message. The replaceAll method searches the message for text matching the regex pattern and replaces it with the replacement string.

#2) Show examples of each: bold, italic, underline, colors (red, green, blue), and combination of bold, italic, underline and a color



Caption (required) ✓

Describe/highlight what's being shown

Showing combination of text formatting

Misc (1 pt.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

Text: Add the pull request link for the branch

Details:

Note: the link should end with /pull/#

URL #1

https://github.com/jah89/jah89-IT114-450/pull/15

^COLLAPSE^

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

Most of the issues I ran into during the milestone was just the syntax. Such as for processing the text and changing to special characters I had to do alot of background research on stackoverflow to get an understanding on it.

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

Files

1 hr 21 mins	Project/Client/Client.java
1 hr	Project/Server/Room.java
23 mins	Project/Server/Server.java
16 mins	…ct/Server/ServerThread.java
15 mins	Project/Common/Payload.java
13 mins	…erver/BaseServerThread.java
6 mins	run.sh
5 mins	Project/Payload.java
4 mins	…ect/Common/PayloadType.java
2 mins	Project/common/LoggerUtil.java

Branches

2 hrs 2 mins	main
1 hr 57 mins	Milestone2
32 secs	Unknown

Showing time spent

Projects • jah89-IT114-450

4 hrs over the Last 7 Days in jah89-IT114-450 under all branches.

Showing time spent

End of Assignment