Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-milestone-2-chatroom-2024-m24/grade/yh68

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

Submissions:

Submission Selection

1 Submission [active] 7/5/2024 7:11:05 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

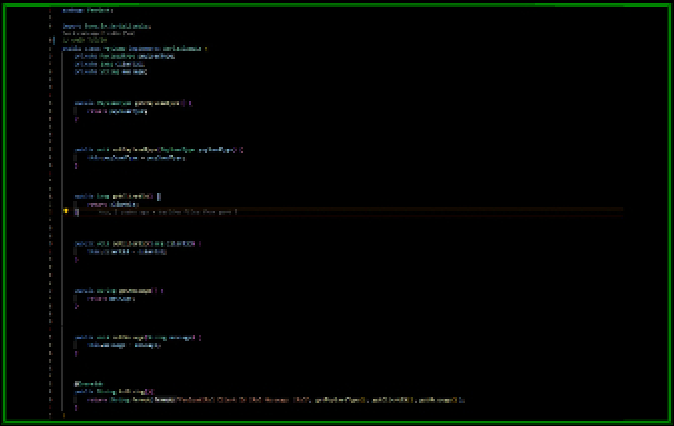
● COLLAPSE

Task #1 - Points: 1

Text: Base Payload Class

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

#1) Show screenshot of the Payload.java



Caption (required) ✓

Describe/highlight what's being shown

Showing a screenshot of the Payload.java class

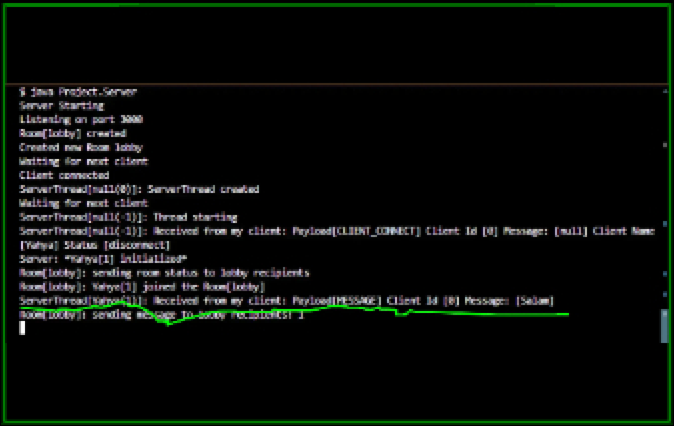
Explanation (required) ✓

Briefly explain the purpose of each property and serialization

PREVIEW RESPONSE

PayloadType payloadType defines the type of payload being sent.
long clientId holds the unique identifier for the client sending the payload.
String message contains the actual message or data being sent with the payload.
By implementing Serializable the Payload class is converted so it can be sent over a network or saved to a file. When it is received it can be deserialized to reconstruct the original Payload object.

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



```
$ java Project.Server
Server Starting
Listening on port 3000
Room[lobby] created
Created new Room lobby
Waiting for next client
Client connected
ServerThread[null(0)]: ServerThread created
Waiting for next client
ServerThread[null(-1)]: Thread starting
ServerThread[null(-1)]: Received from my client: Payload[CLIENT_CONNECT] Client Id [0] Message: [null] Client Name
[Yahya] Status [disconnect]
Server: *Yahya[1] initialized*
Room[lobby]: sending room status to lobby recipients
Room[lobby]: Yahya[1] joined the Room[lobby]
ServerThread[Yahya(1)]: Received from my client: Payload[MESSAGE] Client Id [0] Message: [Salam]
Room[lobby]: sending message to lobby recipients ]
```

Caption (required) ✓

Describe/highlight what's being shown

Showing screenshot examples of the terminal output for base Payload objects

^COLLAPSE ^

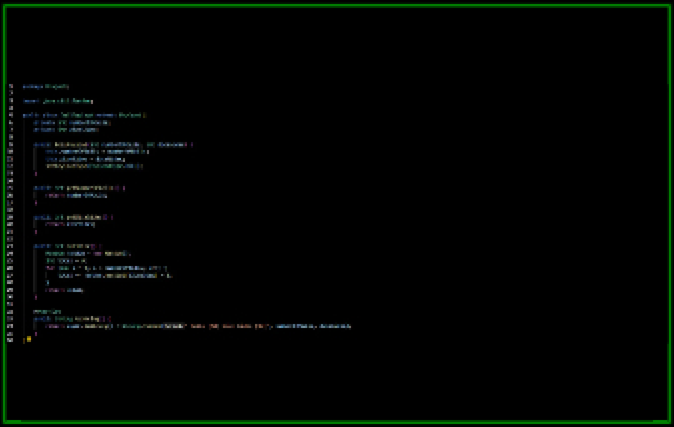
Task #2 - Points: 1

Text: RollPayload Class

Details:

All code screenshots must have ucid/date visible.

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



```
[illegible]
```

Caption (required) ✓

Describe/highlight what's being shown

Showing screenshot of the RollPayload.java class

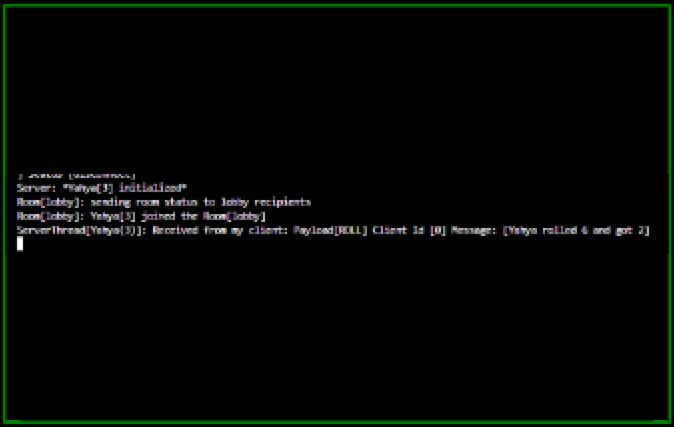
Explanation (required) ✓

Briefly explain the purpose of each property

PREVIEW RESPONSE

numberOfRolls stores the number of times the dice will be rolled.
diceSides represents the number of sides each dice has.
rollDice() method uses a loop to generate random numbers within the specified range and accumulates these values to compute the total sum of the simulated dice rolls. It generates random numbers within the range of 1 to diceSides for each roll specified by numberOfRolls.

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



```
[illegible]
Server: "Yahya[3] initialized"
Room[lobby]: sending room status to lobby recipients
Room[lobby]: Yahya[3] joined the Room[lobby]
ServerThread[Yahya[3]]: Received from my client: Payload[ROLL] Client Id [0] Message: [Yahya rolled 6 and got 2]
█
```

Caption (required) ✓

Describe/highlight what's being shown

Showing screenshot example of the terminal output for base RollPayload objects

● 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

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

Caption (required) ✓
Describe/highlight what's being shown
Showing client side code for handling /roll #

Explanation (required) ✓
Briefly explain the logic
PREVIEW RESPONSE
The method processRollCommand() initializes singleMatcher and uses a predefined regular expression rollSingle. If singleMatcher.matches() returns true, it means the text matches the pattern for a single roll and gets the maximum dice value.

#2) Show the output of a few

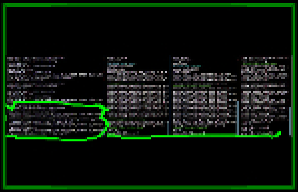
Caption (required) ✓
Describe/highlight what's being shown
Showing the output of a few examples of /roll #

#3) Show the client side code for

Caption (required) ✓
Describe/highlight what's being shown
Showing the client side code for handling /roll #d#

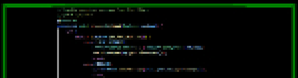
Explanation (required) ✓
Briefly explain the logic
PREVIEW RESPONSE
The method processRollCommand() initializes multiMatcher and uses a predefined regular expression rollMulti. If multiMatcher.matches() returns true, it means the text matches the pattern for a multi roll and gets the number of rolls from the first capture group (multiMatcher.group(1)) and the number of sides on each die from the second capture group (multiMatcher.group(2)).

#4) Show the output of a few

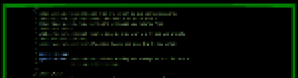


Caption (required) ✓
Describe/highlight what's being shown
Showing the output of a few examples of /roll #d#

#5) Show the ServerThread code



#6) Show the Room code that



Caption (required) ✓

Describe/highlight what's being shown

Showing the ServerThread code receiving the RollPayload

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

When a ROLL payload is received, it processes the payload to get the message containing the result of the dice roll. It calls currentRoom.sendMessage payload.getMessage()); to broadcast the message to all clients in the current room. This message includes the result of the dice roll, so all clients in the room are informed about the outcome.

Caption (required) ✓

Describe/highlight what's being shown

Showing the Room code that processes both Rolls and sends the response

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

Sends a basic String message from the sender to all connectedClients then calls processCommand and evaluates.

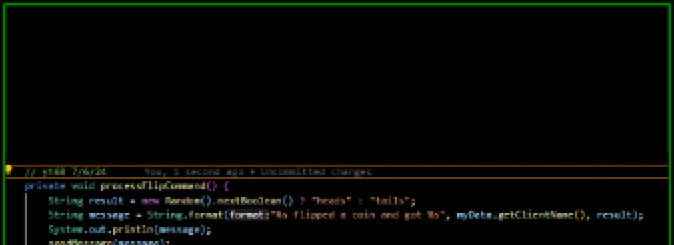
●

COLLAPSE

Task #2 - Points: 1

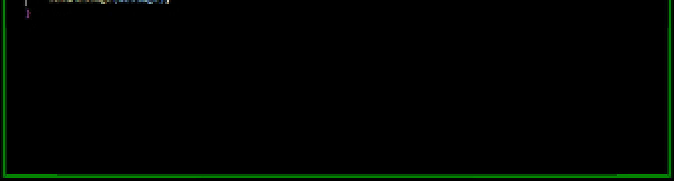
Text: Flip Command

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



```
// yt68 7/6/24       You, 1 second ago • Uncommitted changes
private void processFlipCommand() {
    String result = new Random().nextBoolean() ? "heads" : "tails";
    String message = String.format(format:"%s flipped a coin and got %s", myData.getClientName(), result);
    System.out.println(message);
    sendMessage(message);
```

#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 the client side code for handling /flip

Caption (required) ✓

Describe/highlight what's being shown

Showing the output of a few examples of /flip

Explanation (required) ✓

Briefly explain the logic

PREVIEW RESPONSE

The processFlipCommand method simulates a coin flip for the client. It randomly chooses "heads" or "tails", creates a message with the result, prints it out, and sends the message to the server. Essentially, it lets the client flip a virtual coin and share the outcome with everyone else.

● Text Formatting (3 pts.)

^COLLAPSE^

● ^COLLAPSE^

Task #1 - Points: 1

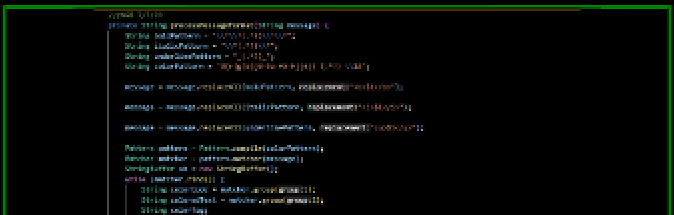
Text: Text Formatting

Details:

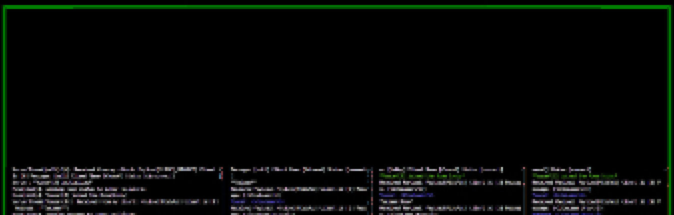
All code screenshots must have ucid/date visible.
Any output screenshots must have at least 3 connected clients able to see the output.
Note: Having the user type out html tags is not valid for this feature, instead treat it like WhatsApp, Discord, Markdown, etc
Note: Each text trigger must wrap the 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.

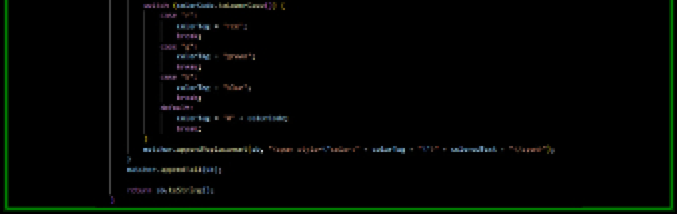
#1) Show the code related to processing the special characters for bold, italic, underline, and colors, and converting them

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



```
//[illegible]
private String processMessageFormat(String message) {
    String boldPattern = "[illegible]";
    String italicPattern = "[illegible]";
    String underlinePattern = "[illegible]";
    String colorPattern = "[illegible]";

    message = message.replaceAll(boldPattern, replacement: "[illegible]");

    message = message.replaceAll(italicPattern, replacement: "[illegible]");

    message = message.replaceAll(underlinePattern, replacement: "[illegible]");

    Pattern pattern = Pattern.compile(colorPattern);
    Matcher matcher = pattern.matcher(message);
    StringBuffer sb = new StringBuffer();
    while (matcher.find()) {
        String colorCode = matcher.group(group: 1);
        String coloredText = matcher.group(group: 2);
        String colorTag;
```





```
            switch (colorCode.toLowerCase()) {
                case "r":
                    colorTag = "red";
                    break;
                case "g":
                    colorTag = "green";
                    break;
                case "b":
                    colorTag = "blue";
                    break;
                default:
                    colorTag = "#" + colorCode;
                    break;
            }
            matcher.appendReplacement(sb, "<span style=\"color:" + colorTag + "\">" + coloredText + "</span>");
        }
        matcher.appendTail(sb);

        return sb.toString();
    }
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to processing the special characters for bold, italic, underline, and colors, and converting them

Explanation (required) ✓

Briefly explain how it works and the choices of the placeholder characters and the result characters

PREVIEW RESPONSE

The method processMessageFormat() that takes a message string and formats it. The expression "#(r|g|b|[0-9a-fA-F]{6}) (.*?) \1#" is used to match color codes in the format #r, #g, #b for predefined colors (red, green, blue) or a 6 character code #FF0000 for custom colors. When the switch statement gets a color code, it takes the specific color code using matcher.group(1). When the matcher iterates through the message string, each matched color code and its associated text are replaced and appended to a StringBuffer (sb).

Caption (required) ✓

Describe/highlight what's being shown

Showing examples of bold, italic, underline, and colors with combination of all

● 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/FreePalestine7/yh68-it114-450/pull/15

● COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I've learned so much doing milestone 2. I think that because the professor made us do this milestone on our own it really helped me grasp a lot of the methods and functions more firmly. I learned about matcher objects as well as pattern objects. While I'm not the best at it yet, I have watched tutorials to help me better understand. I'm starting to better understand string buffers and the concept of payloads.

● ^COLLAPSE ^

Task #3 - Points: 1

Text: WakaTime Screenshot

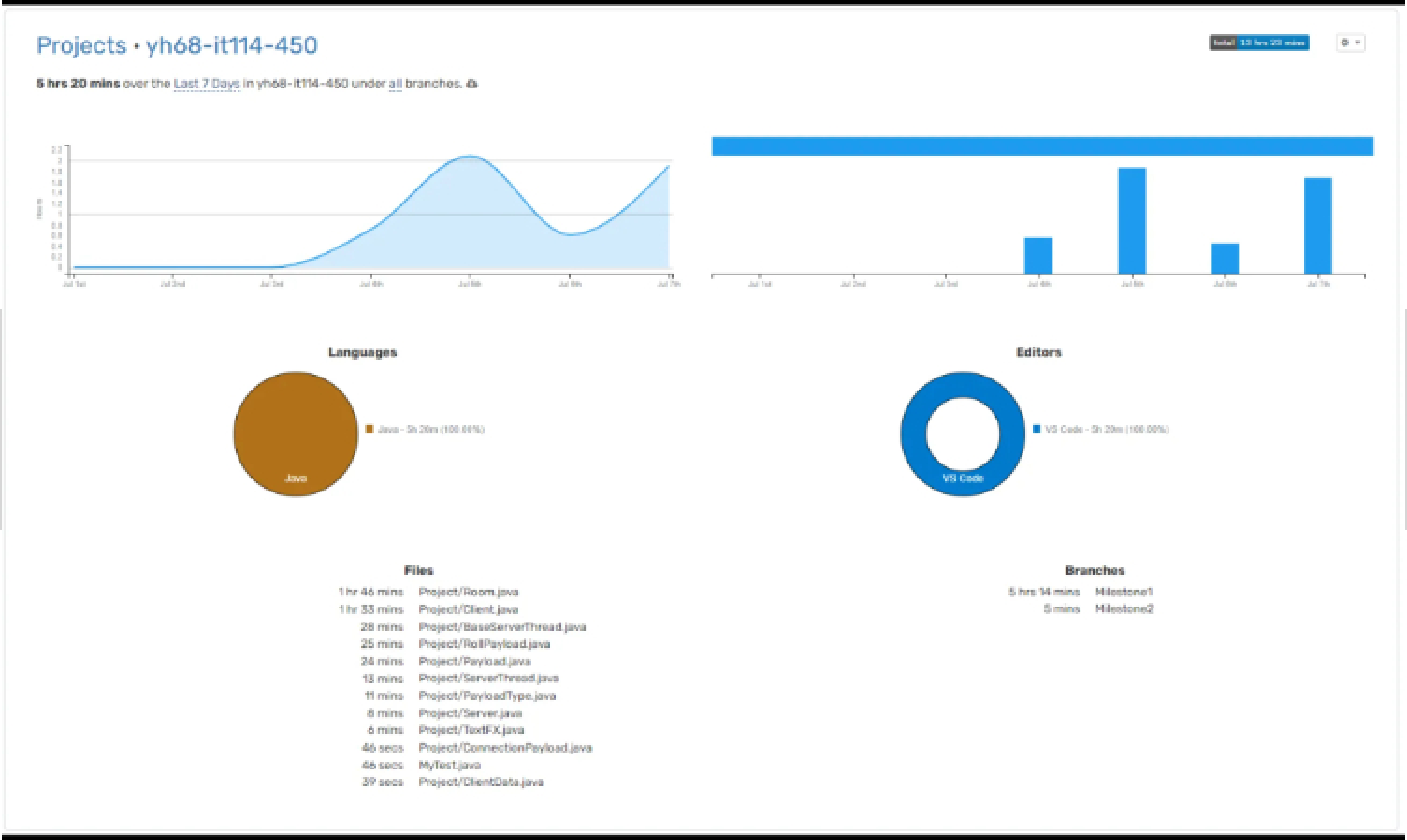
ⓘ Details:

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



Projects • yh68-it114-450

5 hrs 20 mins over the Last 7 Days in yh68-it114-450 under all branches.

■ Java - 5h 20m (100.00%)

■ VS Code - 5h 20m (100.00%)

Files

1 hr 46 mins	Project/Room.java
1 hr 33 mins	Project/Client.java
28 mins	Project/BaseServerThread.java
25 mins	Project/RollPayload.java
24 mins	Project/Payload.java
13 mins	Project/ServerThread.java
11 mins	Project/PayloadType.java
8 mins	Project/Server.java
6 mins	Project/TextFX.java
46 secs	Project/ConnectionPayload.java
46 secs	MyTest.java
39 secs	Project/ClientData.java

Branches

5 hrs 14 mins	Milestone1
5 mins	Milestone2

Wakatime overview (It says Milestone1 because I didn't think to switch branches until it was time to add and commit my changes so I used git stash and just switched branches right before i submitted)

End of Assignment