Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-2-chatroom-2024-m24/grade/el286

Course: IT114-005-F2024
Assigment: [IT114] Milestone 2 Chatroom 2024 (M24)
Student: Erik L. (el286)

Submissions:

Submission Selection

1 Submission [submitted] 11/29/2024 3:59:17 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

Group

100%

Group: Payloads
Tasks: 2
Points: 2

COLLAPSE

Task

100%

Group: Payloads
Task #1: Base Payload Class
Weight: ~50%
Points: ~1.00

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: Payloads
Task #1: Base Payload Class
Sub Task #1: Show screenshot of the Payload.java

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Payload.java Screenshot

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

Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

The PayloadType represents the specific type that will be taken such as ROLL, MESSAGE,FLIP, etc. It tells the server and client what payload is be processed and to be further handled. The clientId is basically the identifier of the client that's sending the payload. Each client is different with its different id. It determines the actions and messages of the specific client. The message simply takes the data that correlates with the payload and is used for normal messaging in the chat. Serialization is used in the Payload to have a communication between the Client and Server. It doens't have to manualy convert and can simply conver the Payload object to a way where it can be communicated over towards the server and the server will deserialize it for processing.

Sub-Task

Group: Payloads

100%

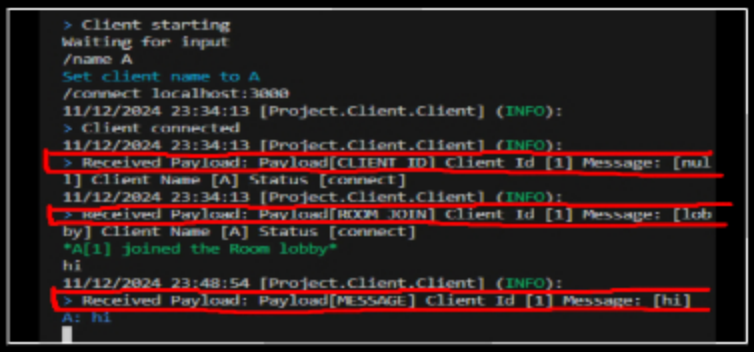
Task #1: Base Payload Class

Sub Task #2: Show screenshot examples of the terminal output for base Payload objects

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Terminal Output for base Payload objects

Caption(s) (required) ✓

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

End of Task 1

Task

100%

Group: Payloads

Task #2: RollPayload Class

Weight: ~50%

Points: ~1.00

COLLAPSE

Details:

All code screenshots must have ucid/date visible.

Columns: 1

Sub-Task

100%

Group: Payloads

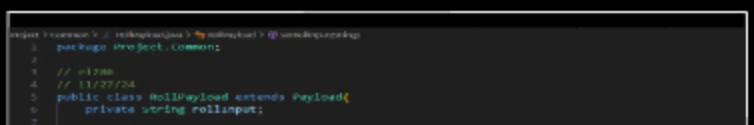
Task #2: RollPayload Class

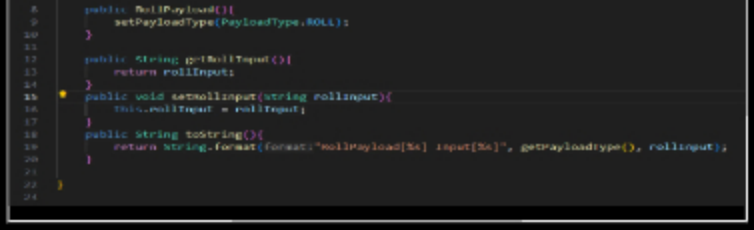
Sub Task #1: Show screenshot of the RollPayload.java (or equivalent)

Task Screenshots

Gallery Style: 2 Columns

4 2 1





screenshot of the RollPayload.java

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the purpose of each property

Response:

Inside the Roll constructor, it basically sets the payload to the enum of ROLL which specifies its exact payload to handle the command for the dice. "rollInput" is declared so it will later store the roll command that will be inputed by the client. It will show eiher (# ex. "2" or #d# ex. "2d4"). Just like in payload it will have its get and setters and toString which will represent the instance of the RollPayload and shown in the server. I added getPayloadType() inside the RollPayload to demonstrate the kind of payload that's used just like how Payload would have it keep it simple.

Sub-Task

100%

Group: Payloads

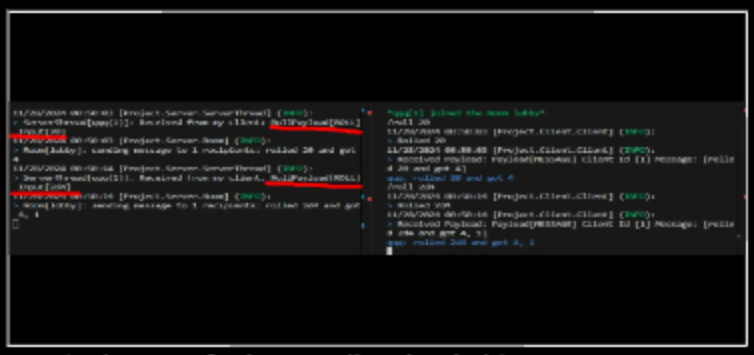
Task #2: RollPayload Class

Sub Task #2: Show screenshot examples of the terminal output for base RollPayload objects

Task Screenshots

Gallery Style: 2 Columns

4 2 1



terminal output for base RollPayload objects

Caption(s) (required) ✓

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

End of Task 2

End of Group: Payloads

Task Status: 2/2

Group

Group: Client Commands

Tasks: 2

100%

Points: 4

COLLAPSE

Task

100%

Group: Client Commands
Task #1: Roll Command
Weight: ~50%
Points: ~2.00

COLLAPSE

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.

Columns: 1

Sub-Task

100%

Group: Client Commands
Task #1: Roll Command
Sub Task #1: Show the client side code for handling /roll #

Task Screenshots

Gallery Style: 2 Columns

4 2 1

client side code for handling /roll #

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

It makes sure that the commanValue "/roll" is not empty and will proceed with the following logic of creating an instance of the RollPayload and its value from the command will be passed by using setRollInput. To send it to the payload, "send" method is used containing roll command from the RollPayload instance created. Used LoggerUtil to log the a message of a successful roll and a not succesful one.

Sub-Task 100%

Group: Client Commands
Task #1: Roll Command
Sub Task #2: Show the output of a few examples of /roll # (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

output of a few examples of /roll #

Caption(s) (required) ✓

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

Sub-Task 100%

Group: Client Commands
Task #1: Roll Command
Sub Task #3: Show the client side code for handling /roll #d# (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

client side code for handling /roll #d#

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

The same logic for the client side is for both roll# and roll #d# which I stated above already: It makes sure that the commanValue "/roll" is not empty and will proceed with the following logic of creating an instance of the RollPayload

and its value from the command will be passed by using setRollInput. To send it to the payload, "send" method is used containing roll command from the RollPayload instance created. Used LoggerUtil to log the a message of a successful roll and a not succesful one.

Sub-Task

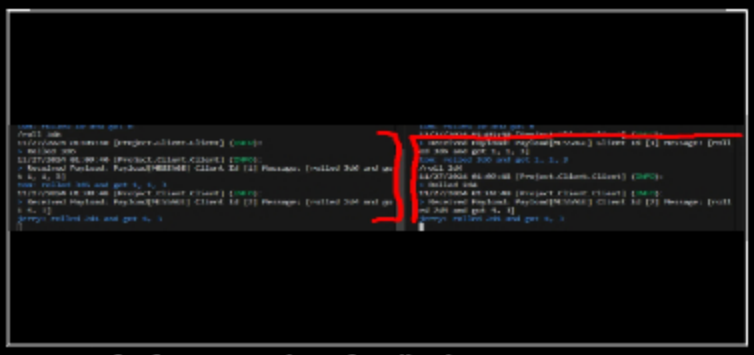
100%

Group: Client Commands
Task #1: Roll Command
Sub Task #4: Show the output of a few examples of /roll #d#

Task Screenshots

Gallery Style: 2 Columns

4 2 1



output of a few examples of /roll #d#

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Client Commands
Task #1: Roll Command
Sub Task #5: Show the ServerThread code receiving the RollPayload

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
public class ServerThread extends BaseServerThread {
    protected void processPayload(Payload payload) {
            // v1200
            // 11/27/24
            case ROLL:
                if(payload instanceof RollPayload){
                    RollPayload rollPayload = (RollPayload) payload;
                    if(currentRoom != null){
                        currentRoom.handleRollCommand(this, rollPayload.getRollInput());
                    }else {
                        sendMessage(message:"Join room to do roll command.");
                    }
                }
                break;
```

ServerThread code receiving the RollPayload

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Briefly explain the logic

Response:

The payload should be meant to be processed for the roll commands and making sure that it's an instance of RollPayload with an if-else does the job. Casted the payload base type of the Payload object to be a RollPayload object. The reason for this is that it can have acces to methods like getRollInput which is from RollPayload. It will retrieve the roll command. Could've gone along with just putting currentRoom.handleRollCommand etc. but to make sure that the client is in the room and not null, an if-else is peformed to verify that the client is in the room and is able to go on forward to handling the roll command and display output to other clients.

Sub-Task

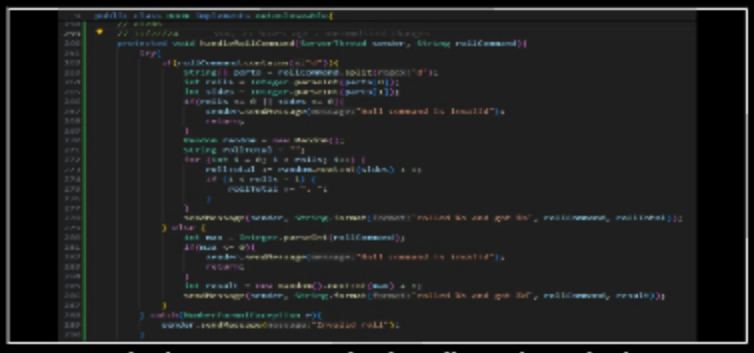
100%

Group: Client Commands
Task #1: Roll Command
Sub Task #6: Show the Room code that processes both Rolls and sends the response

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Room code that processes both Rolls and sends the response

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

Used a try-catch statement to catch a NumberFormatException based on the logic inside the try. First we check if the roll command cotains "d" in the format of the commond and will be split in parts in an array. The first part is the number of dice to roll parts[0] and parts[1] is the sides of the die. Those values are then converted to integers using Integer.parseInt. It's later checked to see that the number is above 0 and not negatives which will send a message saying that the roll command is invalid if not above 0. Random instance is used to simply get random numbers. rollTotal is initialized with an empty string which will alter store the results of the rolls from the for-loop and be appended. The for-loop will iterate a certain amount of times based on the rolls which is the first number in the format of (#d#). rollTotal will be assigned to a new value from the random number of the sides. A "+ 1" is added in order to make the sides range to be fully inclusive. If sides was a 3 it will be (0,1,2,) but by adding + 1, it will include 3 and be 1-3. The statement of "if (i < rolls - 1) { rollTotal += ", "; }" could've been "if (i < rolls)" which will leave an output of ex.(3,4,5,) which leave an extra comma at the end of the last number. For preference, subtracting 1 leaves the last comma out always. It will then send a message using the send(). Now for just getting a single number, max is represented as the single value of the roll and is then converted to an integer. Again if it's below 0 it will send a message about being invalid. Then it will process a random number between 1 and the max value and send the

output using send().

End of Task 1

Task

100%

Group: Client Commands
Task #2: Flip Command
Weight: ~50%
Points: ~2.00

COLLAPSE

Columns: 1

Sub-Task

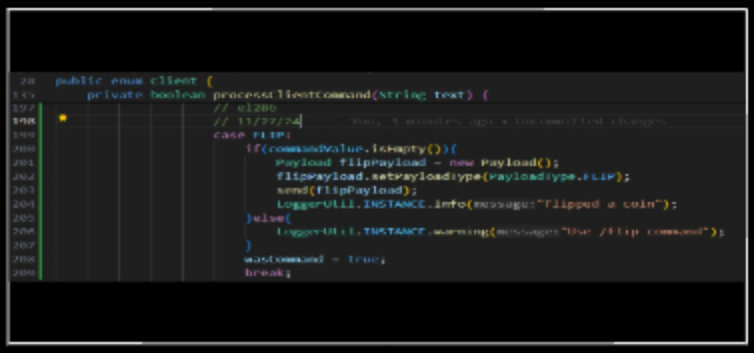
100%

Group: Client Commands
Task #2: Flip Command
Sub Task #1: Show the client side code for handling /flip

Task Screenshots

Gallery Style: 2 Columns

4 2 1



client side code for handling /flip

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

I didn't create another class for the flip and simply used a new Payload instance to indicate the action of the flip. The payload type is set to FLIP which is been added on the class itself which shows that it's requesting for a flip. The flipPayload is being sent byt the send() to the server saying that the client wasnt to do a flip action. LoggerUtil is used to log the message of a succesfful flip or an invalid one.

Sub-Task

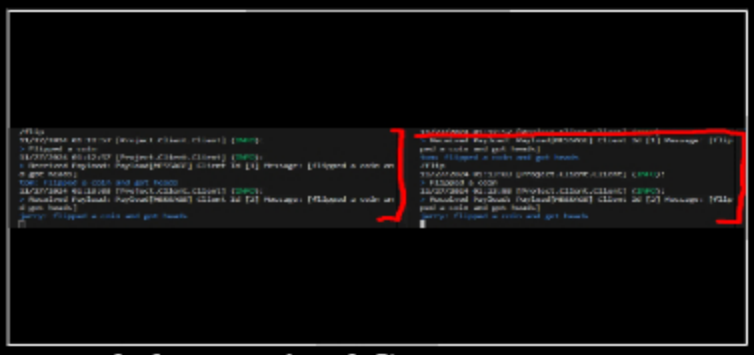
100%

Group: Client Commands
Task #2: Flip Command
Sub Task #2: Show the output of a few examples of /flip (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



output of a few examples of /flip

Caption(s) (required) ✓

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

End of Task 2

End of Group: Client Commands
Task Status: 2/2

Group

100%

Group: Text Formatting
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Text Formatting
Task #1: Text Formatting
Weight: ~100%
Points: ~3.00

COLLAPSE

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

Columns: 1

Sub-Task

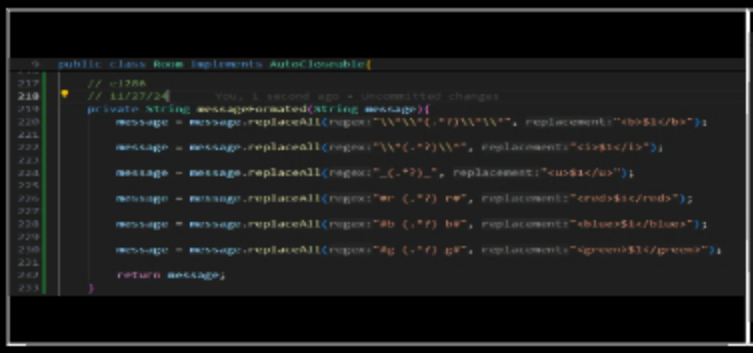
100%

Group: Text Formatting
Task #1: Text Formatting
Sub Task #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)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



code related to processing the special characters

code related to send the message with the respective format

Caption(s) (required) ✓

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

Task Response Prompt

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

Response:

messageFormatted method is created and takes a message which is a string that follows a specific fomat. The message are intialized and later reinitalized with other formats. The message usees replaceAll() which takes the regex and then replaces with a string. For bold its literal will be ** and and the (.*?) gets any character between the ** and then closes with another pair of **. This concept goes for about the same with the others just different literals of course. These placeholders are use to specify the part of the message that's going to fit the followig foramt. It follows a similar style like in the examples provided in the doc. These characters are then replaced by html tags like , , etc. in the output. In order for all this to work, I edited inside the sendMessage() where I created a String that holds the output of the method messageFormated and replaced the "message" that was getting before with the new "messageFormat" that contains the formating.

Sub-Task

100%

Group: Text Formatting

Task #1: Text Formatting

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

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

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

End of Task 1

End of Group: Text Formatting
Task Status: 1/1

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%
Points: ~0.33

COLLAPSE

Details:
Note: the link should end with /pull/#

Task URLs

URL #1
https://github.com/ElLopez21/el286-IT114-005/pull/11

URL
https://github.com/ElLopez21/el286-IT114-005/p

End of Task 1

Task

100%

Group: Misc
Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Task Response Prompt

Response:

Getting the payload to work in which it was trying to register for the communication of both clients was trouble. It wasn't registering on the other clients the result and just on one client only. At first the roll for the format of #d# was somewhat working where it would give me random value but not the amount of values based on the roll so if put 3d6, it would only display 1 output instead of three like 2,2,4. I made adjustments to it based on how I did on the earlier homeworks for flip and roll. The message format was also difficult in which I had to search for the regex pattern correctly for each format.

End of Task 2

Task

100%

Group: Misc
Task #3: WakaTime Screenshot
Weight: ~33%
Points: ~0.33

COLLAPSE

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 Screenshot

WakaTime Screenshot

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment