

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-2-chatroom-2024-m24/grade/mcp62>

Course: IT114-003-F2024

Assignment: [IT114] Milestone 2 Chatroom 2024 (M24)

Student: Michael P. (mcp62)

Submissions:

Submission Selection

1 Submission [submitted] 11/19/2024 7:50:18 PM

Instructions

^ COLLAPSE ^

1. Implement the Milestone 2 features from the project's proposal document:
<https://docs.google.com/document/d/1ONmvEveI97GTFPGfVwwQC96xSsobbSbk56145XizQG4/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



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



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

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 payload.java class encapsulate a structured message that can be transmitted between systems or components in a serialized format. It holds three key pieces of data: a PayloadType (which likely categorizes the type of the message), a clientId (which identifies the client associated with the payload), and a message (which contains the actual content or data being sent). The class implements the Serializable interface, allowing it to be easily serialized for storage or network communication.

Sub-Task

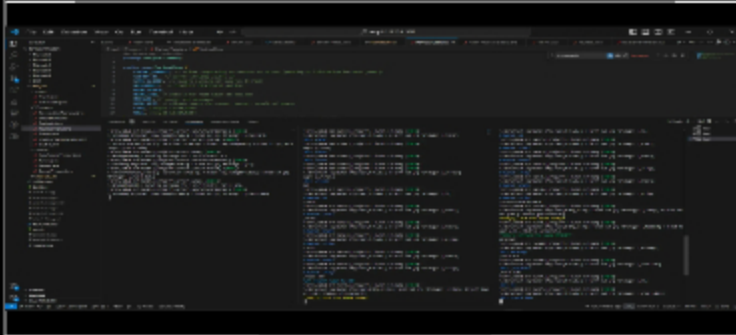


Group: Payloads
Task #1: Base Payload Class
Sub Task #2: Show screenshot examples of the terminal output for base Payload objects

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows message, roll, flip, and create room

Caption(s) (required) ✓

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

End of Task 1

Task



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



Group: Payloads

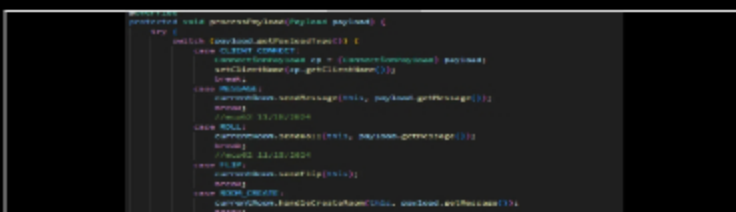
Task #2: RollPayload Class

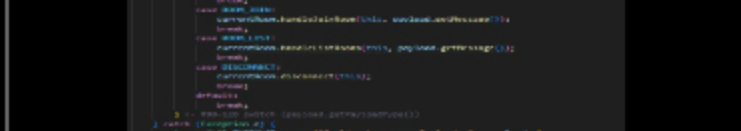
Sub Task #1: Show screenshot of the RollPayload.java (or equivalent)

Task Screenshots

Gallery Style: 2 Columns

4 2 1





ROLL and FLIP case

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the purpose of each property

Response:

the client creates a payload of type ROLL, which sends either a single number or two numbers separated by a space, both as strings. This payload is sent to the ServerThread, which recognizes it as type ROLL and forwards it to the roll() method of the Room class. The Room class then determines whether the payload contains one number or two. If it contains one number, a random value between 1 and that number is generated. If it contains two numbers, a dice roll is performed. The result is then passed to the sendMessage() method of the Room class, which sends it back to the ServerThread, and the result is finally broadcast to all clients.

Sub-Task

Group: Payloads

Task #2: RollPayload Class

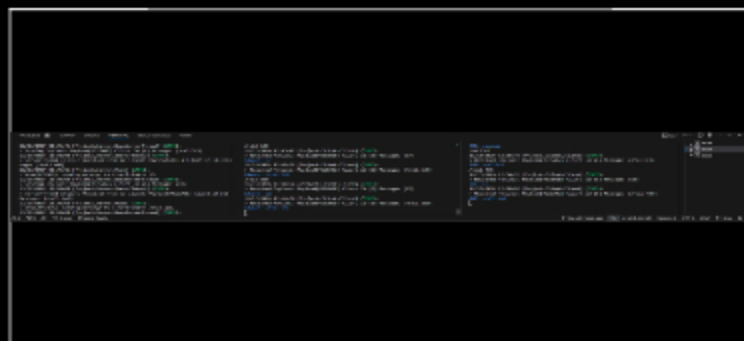
Sub Task #2: Show screenshot examples of the terminal output for base RollPayload objects

100%

Task Screenshots

Gallery Style: 2 Columns

4 2 1



roll payload in terminal

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

Points: 4

100%

^ COLLAPSE ^

Task



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

^ COLLAPSE ^

Details:

All code screenshots must have uid/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: 4

Sub-Task



Group:
Client
Commands
Task #1:
Roll
Command
Sub Task

Sub-Task



Group:
Client
Commands
Task #1:
Roll
Command
Sub Task

Sub-Task



Group:
Client
Commands
Task #1:
Roll
Command
Sub Task

Sub-Task



Group:
Client
Commands
Task #1:
Roll
Command
Sub Task



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



client side
code for roll

Caption(s) (required) ✓

Caption Hint:
Describe/highlight what's
being shown



Task

Response

Prompt

Briefly explain the logic

Response:



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



/roll # output

Caption(s) (required) ✓

Caption Hint:
Describe/highlight what's
being shown



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



How i
handle logic
with roll
parsing

Caption(s) (required) ✓

Caption Hint:
Describe/highlight what's
being shown



Task

Response

Prompt

Briefly explain the logic



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



Payload.java
/roll #d#
output

Caption(s) (required) ✓

Caption Hint:
Describe/highlight what's
being shown

Response:

On the client side the message is detected if the message payload contains /roll if so a new payload is made and that is set to type payload which then gets sent off.

The sendRoll method is a synchronized method that processes a roll command from a client and sends the result back. It starts by stripping off the first six characters from the message (presumably to remove a command identifier like "ROLL "). The method then checks if the message contains the character "d", indicating a dice roll in the format XdY (where X is the number of dice, and Y is the number of sides per die). If "d" is present, the message is split into two parts: X (the number of dice) and Y (the sides of each die). It iterates X times, generating a random number between 1 and Y for each die roll and summing the results. If "d" is not present, the method interprets the message as a single number, generating a random value between 1 and that number. The final result is then converted to a string and sent back to the client using the sendMessage method, ensuring the client receives the calculated roll result.

Sub-Task


100%


Group:
Client
Commands
Task #1:
Roll
Command
Sub Task

Sub-Task

100%

Group:
Client
Commands
Task #1:
Roll
Command
Sub Task

 Task
Screenshots

 Task
Screenshots

4 2 1



/roll handled
in server
thread as a
case
statement

Caption(s) (required) ✓

Caption Hint:

*Describe/highlight what's
being shown*

⇒ Task

Response

Prompt

Briefly explain the logic

Response:

In the ServerThread class, the ROLL case handles dice roll commands from clients. When a payload with type ROLL is received, the sendRoll method of the currentRoom is called, passing the ServerThread instance (representing the sender) and the message from the payload. The sendRoll method processes the roll logic—determining if it's a single number or a dice roll, generating the appropriate random result, and summing them if necessary. It then sends the computed result back to the sender using the sendMessage method, so that the client receives the outcome of their roll.

4 2 1



again roll logic
and below
sends the
response

Caption(s) (required) ✓

Caption Hint:

*Describe/highlight what's
being shown*

⇒ Task

Response

Prompt

Briefly explain the logic

Response:

In my case I was able to make roll work by passing it as a message and doing all the logic on the server side prior to sending the output back out.

Task



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

^ COLLAPSE ^

Columns: 1

Sub-Task



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

```
//mcp62 11/18/2024
} else if (message.startsWith(prefix: "/flip")){
    Payload p = new Payload();
    p.setPayloadType(PayloadType.FLIP);
    p.setMessage(message);
    send(p);
} <- #254-259 else if (message.startsWith("/flip"))
Payload p = new Payload();
p.setPayloadType(PayloadType.MESSAGE);
p.setMessage(message);
send(p);
<- #246-264 private void sendMessage(String message)
```

/flip logic

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown

≡ Task Response Prompt

Briefly explain the logic

Response:

Flip works simply as a trigger to a message with the prfix /flip, if this is true the block runs and a new paylaod is created, the new payload is set to type FLIP which is then sends off the /flip command to be processed.

Sub-Task

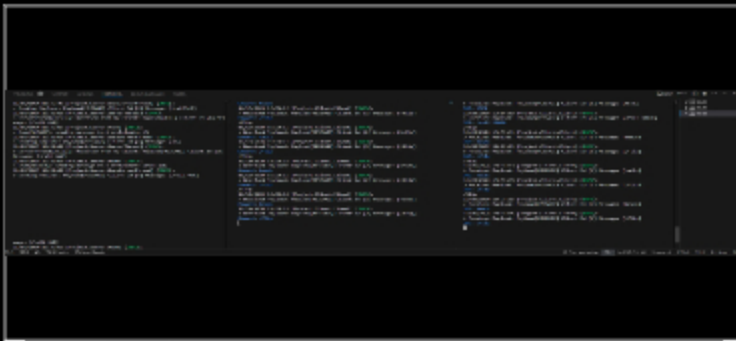


Group: Client Commands
Task #2: Flip Command
Sub Task #2: Show the output of a few examples of /flip (related payload output should be visible)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



/flip output shows heads or tails

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 ^

i 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



special characters statement's using regex

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:

The statment processes a message (string) to replace specific markers (like !b, !r, !_, *, and !) with corresponding tags for formatting the colors I will implement in Milestone 3. The logic operates inside a while loop that repeatedly checks for the presence of these markers in the message. When a marker is found, it uses `replaceFirst` to replace the first occurrence of the marker with an opening tag and another occurrence with a closing tag. As such !b would be replaced with `` and ``. The loop continues until no further markers are found, at which point when a message matches the unchanged `originalMessage` the loop will then exit. This is set up so that all markers in the string are replaced with their proper tags, even if multiple different markers are present.

Sub-Task

100%

Group: Text Formatting

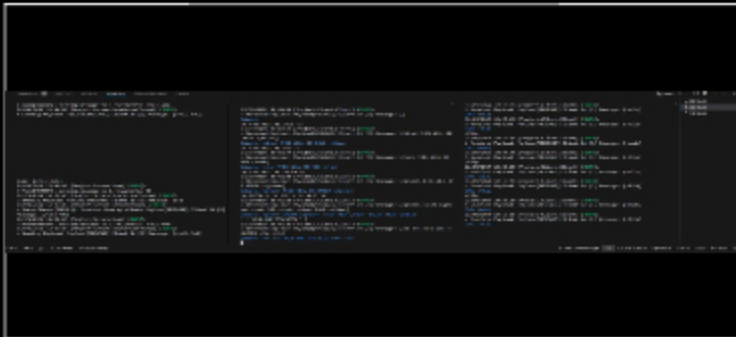
Task #1: Text Formatting

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



Shows examples of each and combinations in single messages (middle client)

Caption(s) (required) ✓

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

End of Group: Text Formatting

Task Status: 1/1

Group



Group: Misc

Tasks: 3

Points: 1

^ COLLAPSE ^

Task



Group: Misc

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

Weight: ~33%

Points: ~0.33

^ COLLAPSE ^

i Details:

Note: the link should end with /pull/#



Task URLs

URL #1

<https://github.com/Onervv/mcp62-IT114-003/pull/15>

URL

<https://github.com/Onervv/mcp62-IT114-003/pul>

End of Task 1

Task



Group: Misc

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

Weight: ~33%

Points: ~0.33

^ COLLAPSE ^

Task Response Prompt

Response:

One thing that you can see through the images was the roll command, for now Im going to leave as is, although it is functionl it is being processed as a String message as opposed to its own payload type. I see where i could refactor this but I dont want to make significant changes until I finish the last Milestone or have more time to work on it. Other issues were with using the regex patterns, my origianl values needed to be escaped otherwise they were reserved characters and caused a weird infinite loop in my while true statement.

End of Task 2

Task



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

Time	Repository
0:00:00	ProtonMail/ProtonMail-Android
0:00:01	ProtonMail/ProtonMail-Android
0:00:02	ProtonMail/ProtonMail-Android
0:00:03	ProtonMail/ProtonMail-Android
0:00:04	ProtonMail/ProtonMail-Android
0:00:05	ProtonMail/ProtonMail-Android
0:00:06	ProtonMail/ProtonMail-Android
0:00:07	ProtonMail/ProtonMail-Android
0:00:08	ProtonMail/ProtonMail-Android
0:00:09	ProtonMail/ProtonMail-Android
0:00:10	ProtonMail/ProtonMail-Android
0:00:11	ProtonMail/ProtonMail-Android
0:00:12	ProtonMail/ProtonMail-Android
0:00:13	ProtonMail/ProtonMail-Android
0:00:14	ProtonMail/ProtonMail-Android
0:00:15	ProtonMail/ProtonMail-Android
0:00:16	ProtonMail/ProtonMail-Android
0:00:17	ProtonMail/ProtonMail-Android
0:00:18	ProtonMail/ProtonMail-Android
0:00:19	ProtonMail/ProtonMail-Android
0:00:20	ProtonMail/ProtonMail-Android
0:00:21	ProtonMail/ProtonMail-Android
0:00:22	ProtonMail/ProtonMail-Android
0:00:23	ProtonMail/ProtonMail-Android
0:00:24	ProtonMail/ProtonMail-Android
0:00:25	ProtonMail/ProtonMail-Android
0:00:26	ProtonMail/ProtonMail-Android
0:00:27	ProtonMail/ProtonMail-Android
0:00:28	ProtonMail/ProtonMail-Android
0:00:29	ProtonMail/ProtonMail-Android
0:00:30	ProtonMail/ProtonMail-Android
0:00:31	ProtonMail/ProtonMail-Android
0:00:32	ProtonMail/ProtonMail-Android
0:00:33	ProtonMail/ProtonMail-Android
0:00:34	ProtonMail/ProtonMail-Android
0:00:35	ProtonMail/ProtonMail-Android
0:00:36	ProtonMail/ProtonMail-Android
0:00:37	ProtonMail/ProtonMail-Android
0:00:38	ProtonMail/ProtonMail-Android
0:00:39	ProtonMail/ProtonMail-Android
0:00:40	ProtonMail/ProtonMail-Android
0:00:41	ProtonMail/ProtonMail-Android
0:00:42	ProtonMail/ProtonMail-Android
0:00:43	ProtonMail/ProtonMail-Android
0:00:44	ProtonMail/ProtonMail-Android
0:00:45	ProtonMail/ProtonMail-Android
0:00:46	ProtonMail/ProtonMail-Android
0:00:47	ProtonMail/ProtonMail-Android
0:00:48	ProtonMail/ProtonMail-Android
0:00:49	ProtonMail/ProtonMail-Android
0:00:50	ProtonMail/ProtonMail-Android
0:00:51	ProtonMail/ProtonMail-Android
0:00:52	ProtonMail/ProtonMail-Android
0:00:53	ProtonMail/ProtonMail-Android
0:00:54	ProtonMail/ProtonMail-Android
0:00:55	ProtonMail/ProtonMail-Android
0:00:56	ProtonMail/ProtonMail-Android
0:00:57	ProtonMail/ProtonMail-Android
0:00:58	ProtonMail/ProtonMail-Android
0:00:59	ProtonMail/ProtonMail-Android
0:01:00	ProtonMail/ProtonMail-Android

WAKA TIME

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment