Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-2-chatroom-2024-m24/grade/rra23

Course: IT114-003-F2024
Assigment: [IT114] Milestone 2 Chatroom 2024 (M24)
Student: Rahid A. (rra23)

Submissions:

Submission Selection

1 Submission [submitted] 11/11/2024 11:44:00 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

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%

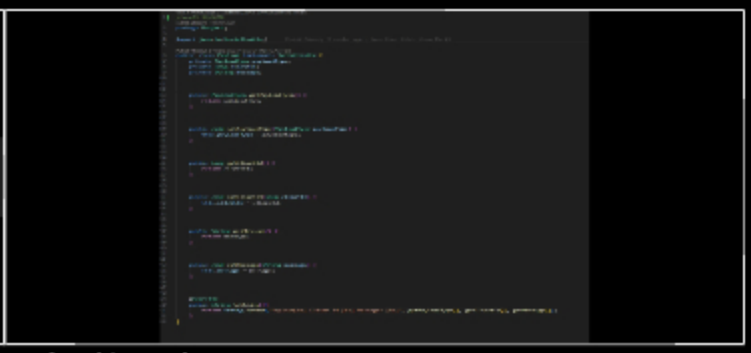
Group: Payloads
Task #1: Base Payload Class
Sub Task #1: Show screenshot of the Payload.java

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Payload.java class



payload.java class

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

Task Response Prompt

Briefly explain the purpose of each property and serialization

Response:

payloadType is the type of payload that is being sent

clientId is the id of the client that is sending the payload

message is the message that is being sent with the payload

Serializable allows the payload to be sent between the server and the client. It converts the payload into a byte stream and sends that. This makes it so that we are not just sending the reference to the payload, but the actual paylod.

Sub-Task

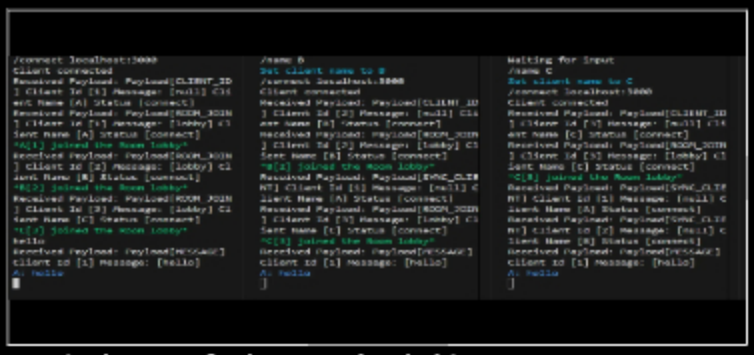
100%

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



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

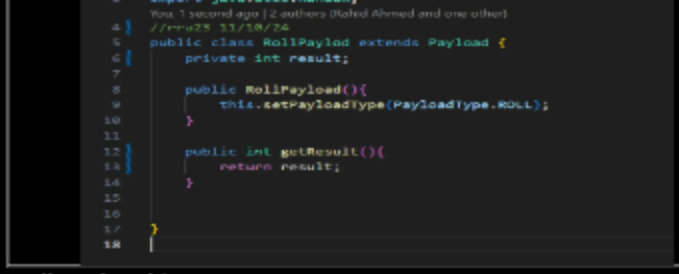
100%

Group: Payloads
Task #2: RollPayload Class
Sub Task #1: Show screenshot of the RollPayload.java (or equivalent)

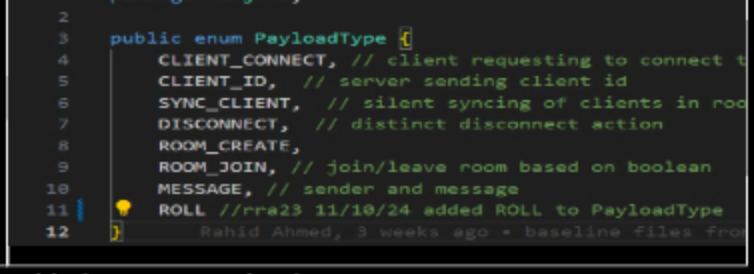
Task Screenshots

Gallery Style: 2 Columns

4 2 1



RollPayload.java



Added ROLL to PayloadType

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the purpose of each property

Response:

The purpose of the result property is to store the results of the dice roll.

Sub-Task

100%

Group: Payloads

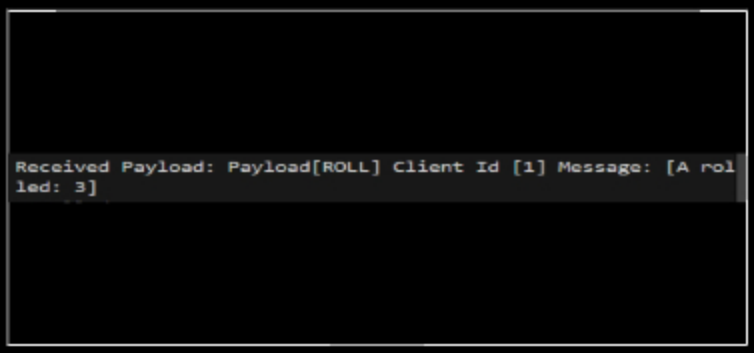
Task #2: RollPayload Class

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



examples of output for roll payload

Caption(s) (required) ✓

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

End of Task 2

End of Group: Payloads

Task Status: 2/2

Group

100%

Group: Client Commands

Tasks: 2

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

```
} else if (text.startsWith("/roll")) {  //rra23 11/10/24
    try{
        String rollString = text.replace("/roll", "").trim();
        sendRoll(rollString);
    } catch (Exception e){
        e.printStackTrace();
    }
    return true;
```

code for processing the /roll command on the client side

```
//rra23 11/10/24
private void sendRoll(String rollString){
    try{
    Payload rp = new Payload();
    rp.setPayloadType(PayloadType.ROLL);
    rp.setMessage(rollString);
    out.writeObject(rp);
    } catch(Exception e){
        e.printStackTrace();
    }
}       You, 9 minutes ago • Uncommitted change
```

code for sending roll payload to the server thread

```
    break;
case PayloadType.ROLL: //rra23 11/11/24
    processMessage(payload.getClientId(),payload.getMessage());
    break;
```

code for output of the roll result

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

In the first screenshot, I added the code to process the command by adding another else if block to the processClientCommands method. This code takes in the input from the client and trims the command so that it is left with just the parameters. Then it calls the roll command.

In the second screenshot, I created and sent the roll payload to the server.

In the third screenshot, I added a case to the processPayload method to display what the client rolled

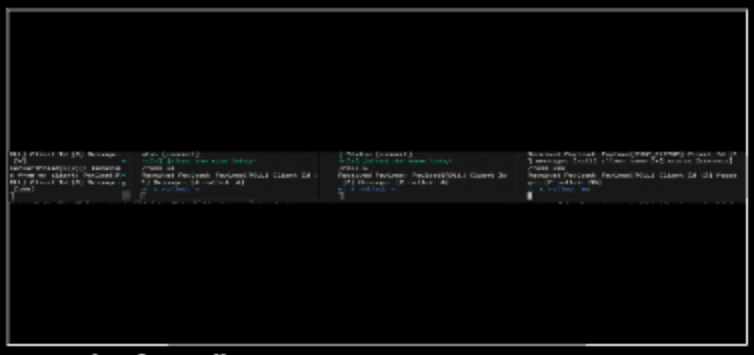
Sub-Task 100%

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



examples for /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

code for processing roll payload received from server

code for processing command on the client side

code for processing roll payload received from server thread

code for processing command on the client side

```
//rra23 11/10/24
private void sendRoll(String rollString){
    try{
    Payload rp = new Payload();
    rp.setPayloadType(PayloadType.ROLL);
    rp.setMessage(rollString);
    out.writeObject(rp);
    } catch(Exception e){
        e.printStackTrace();
    }
```

code for creating and sending the roll payload to the server thread

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

the /roll # and /roll #d# uses the same code on the client side, since I wrote it so that the command in separated on the serverthread side.

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

examples of /roll#d#

Caption(s) (required) ✓

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

Sub-Task 100%

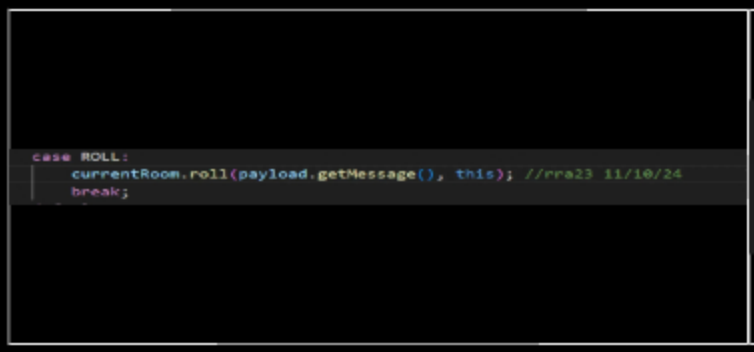
Group: Client Commands
Task #1: Roll Command
Sub Task #5: Show the ServerThread code receiving the RollPayload

Task Screenshots

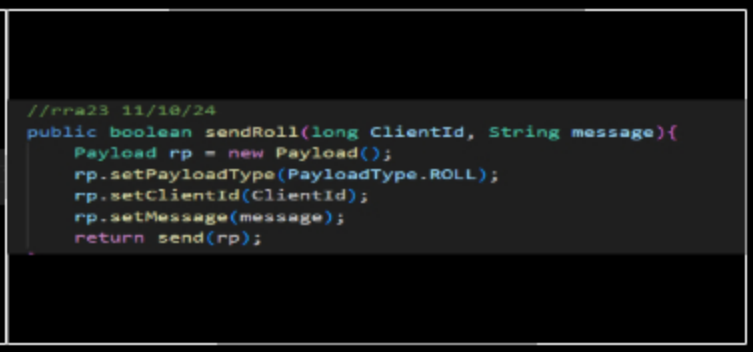
Task Screenshots

Gallery Style: 2 Columns

4 2 1



code for calling the roll method in the current room



code for sending Roll payload back to the client

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

In the first image I added a case for ROLL in the processPayload method, which calls the roll method from the room class

In the second image I created a method called sendRoll which creates a roll payload with the results and sends it back to the client

Sub-Task

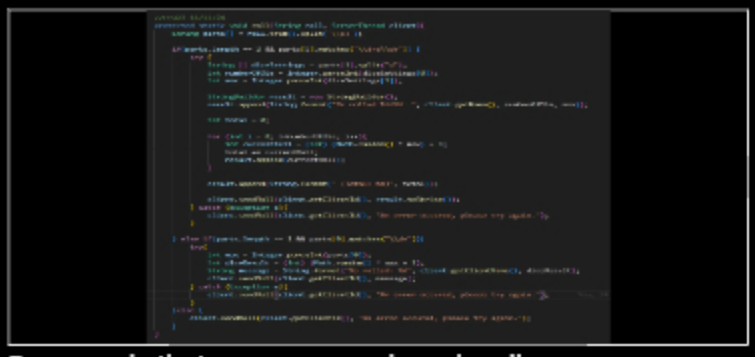
100%

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 and sends rolls

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Briefly explain the logic

Response:

the method takes in the roll command from the payload and the serverthread who sent the payload.

then it splits the rolling parameters depending on if it has the character "d", since that is what determines the difference between one and multiple die.

These parameters are stored in variables which are used with Math.random() in order to generate a random number between 1 and the max entered by the client. And if the user chose to roll more than one die, it goes into a loop and adds each result to the variable total.

Then it sends the roll back to the serverthread, which is then sent back to the client.

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

```
} else if(text.equalsIgnoreCase("/flip")){ //rra23 11/11/24
    try {
        sendFlip();
    } catch (Exception e){
        e.printStackTrace();
    }
    return true;      You, 1 second ago • Uncommitted changes
```

code for processing when the client uses the /flip command

```
//rra23 11/11/24
private void sendFlip(){
    try{
    Payload fp = new Payload();
    fp.setPayloadType(PayloadType.FLIP);
    out.writeObject(fp);
    } catch(Exception e){
        e.printStackTrace();        You, 1
    }

}
```

code for creating and sending the flip payload

```
case PayloadType.FLIP: //rra24 11/11/24
    processMessage(payload.getClientId(), payload.getMessage());
    break;      You, 1 second ago • Uncommitted changes
```

code for displaying results of the flip command

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

Task Response Prompt

Briefly explain the logic
Response:

In the first screenshot I added an else if statement to the processClientCommand method in order to see if the user uses the /flip command

In the second screenshot I added a sendFlip method that sends the flip payload from the client to the serverthread.

In the third picture I added a case for the FLIP payload in the processPayload method in order to display the results of the coin flip.

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

example outputs for /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 for formatting text

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:

I added if statements for each format that the user can choose from. I added it to sendMessage method in Room.java. If the message contains any of the special symbols, it converts the symbols to the text formatting tags. Then it calls the sendMessage method with the new formatted text.

bold would change to **bold**

italic would change to italic

underline would change to underline

(for bold, italic, and underline the text that I am typing into this box is getting formatted, but the way I typed it is how it comes out in my code)

#rred#r would change to red

#bblue#r would change to blue

#ggreen#g would change to green

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 for bold, italic, and underline

red green and blue examples

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

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

100%

Weight: ~33%

Points: ~0.33

COLLAPSE

Details:

Note: the link should end with /pull/#

Task URLs

URL #1

https://github.com/Rahid-Ahmed/rra23-IT114-003/pull/14

URL

https://github.com/Rahid-Ahmed/rra23-IT114-00

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:

I was having a hard time with the text formatting. At first I tried to create methods for each format, but that was not working, so then I tried to change the existing message, which was also not working. Then I struggled trying to get the replace method to work. Bold was the first one I got to work, so I then just copied and pasted what I wrote for bold and changed the characters depending on which format it was.

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 overview

wakatime file view

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment