Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-module-5-project-milestone-1/grade/el286

Course: IT114-005-F2024
Assigment: [IT114] Module 5 Project Milestone 1
Student: Erik L. (el286)

Submissions:

Submission Selection

1 Submission [submitted] 10/28/2024 11:39:30 PM

Instructions

COLLAPSE

Overview Video: https://youtu.be/A2yDMS9TS1o

1. Create a new branch called Milestone1
2. At the root of your repository create a folder called Project if one doesn't exist yet
    1. You will be updating this folder with new code as you do milestones
    2. You won't be creating separate folders for milestones; milestones are just branches
3. Copy in the code from Sockets Part 5 into the Project folder (just the files)
    2. https://github.com/MattToegel/IT114/tree/M24-Sockets-Part5
4. Fix the package references at the top of each file (these are the only edits you should do at this point)
5. Git add/commit the baseline and push it to github
6. Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
7. Ensure the sample is working and fill in the below deliverables 1. Note: Don't forget the client commands are /name and /connect
8. Generate the output file once done and add it to your local repository
9. Git add/commit/push all changes
10. Complete the pull request merge from the step in the beginning
11. Locally checkout main
12. git pull origin main

Branch name: Milestone1

Group

100%

Group: Start Up
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Start Up
Task #1: Start Up
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:
Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 4

Sub-Task

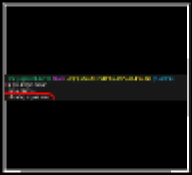
100%

Group: Start Up
Task #1: Start Up
Sub Task #1: Show the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Listening for connections

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

Sub-Task

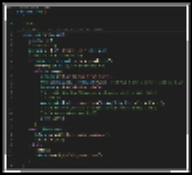
100%

Group: Start Up
Task #1: Start Up
Sub Task #2: Show the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Server code listening for connections

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown (ucid/date must be present)

Task

Sub-Task

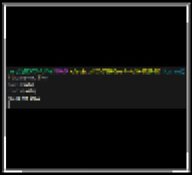
100%

Group: Start Up
Task #1: Start Up
Sub Task #3: Show the Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Client Starting

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

Sub-Task

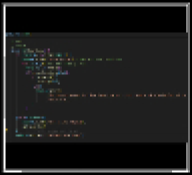
100%

Group: Start Up
Task #1: Start Up
Sub Task #4: Show the Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Client Code that prepares the client and waits for user input

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown (ucid/date must

Response
Prompt

Briefly explain the code related to starting up and waiting for connections

Response:

The start method passes a port integer which is expected to be 3000 in order to connect. It uses ServerSocket for incoming connections and will try to do some unless it fails. If it passes, it'll create the first room and the while it's running, it waits for client connections and will later be accepted which then creates a ServerThread to deal with communication on the client side for each client. There's error handling to see if the connection fails and there's the method "shutdown" in order to disconnect all clients.

be present)

Task
Response
Prompt

Briefly explain the code/logic/flow leading up to and including waiting for user input

Response:

The listenToInput() informs in the console that it's waiting for a input when the client is listening for a user input. This is simply done using Scanner where it reads from the console and later informs the user that client is ready to have some input. It will stil wait for the client if the client is stil active though for a text message.

End of Task 1

Task

100%

Group: Start Up
Task #2: Connecting
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:
Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 2

Sub-Task

Group: Start Up
Task #2: Connecting

Sub-Task

Group: Start Up
Task #2: Connecting

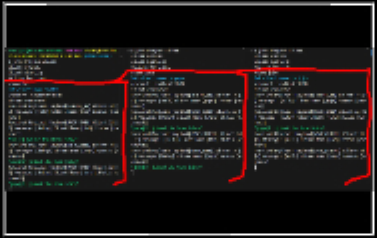
100%

Sub Task #1: Show 3 Clients connecting to the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Clients connected to Server inside the red boxes.

Caption(s) (required) ✓

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

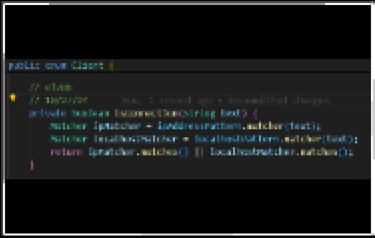
100%

Sub Task #2: Show the code related to Clients connecting to the Server (including the two needed commands)

Task Screenshots

Gallery Style: 2 Columns

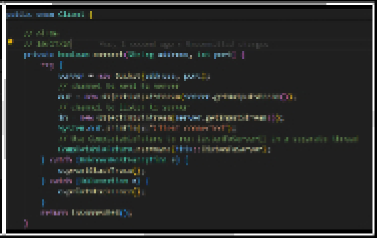
4 2 1



```
public enum Client {

    // [illegible]
    // [illegible]
    private boolean isConnection(String text) {
        Matcher ipMatcher = ipAddressPattern.matcher(text);
        Matcher localhostMatcher = localhostPattern.matcher(text);
        return ipMatcher.matches() || localhostMatcher.matches();
    }
```

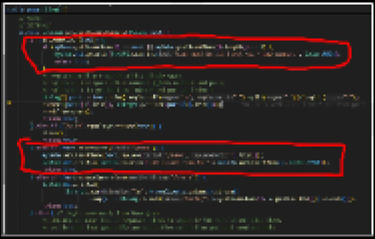
Client connecting to server



```
public enum Client {

    // [illegible]
    // [illegible]
    private boolean connect(String address, int port) {
        try {
            server = new Socket(address, port);
            // channel to send to server
            out = new ObjectOutputStream(server.getOutputStream());
            // channel to listen to server
            in = new ObjectInputStream(server.getInputStream());
            System.out.println("Client connected");
            // Use CompletableFuture to run listenToServer() in a separate thread
            CompletableFuture.runAsync(this::listenToServer);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return isConnected();
    }
```

Client connecting to server



Clients connecting to the Server

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow

Response:

In the processClientCommand, it starts off by making sure that the client has a passing name before connecting and if that fails, it will display the message in red letting the user know the error. If the command is started with /name with a respectful name then it will set it. For isConnection(), it validates the connection command /connect which the method passes the 'String Text'. Once everything passes the connect() method will try to establish a connection to the server.

End of Task 2

End of Group: Start Up
Task Status: 2/2

Group

Group: Communication
Tasks: 2

100%

Tasks: 2

Points: 3

COLLAPSE

Task

100%

Group: Communication
Task #1: Communication
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 4

Sub-Task

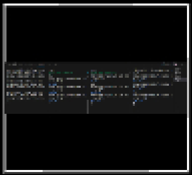
100%

Group: Communicatio
Task #1: Communicatio
Sub Task #1: Show each Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1



sending and receiving messages

Caption(s) (required) ✓

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

Sub-Task

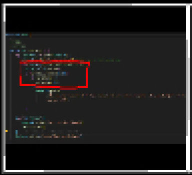
100%

Group: Communicatio
Task #1: Communicatio
Sub Task #2: Show the code

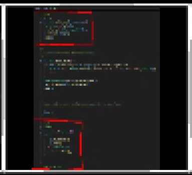
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Client-side of getting a user message



Client-side of getting a user message and sending it over the socket

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task

Sub-Task

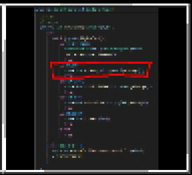
100%

Group: Communicatio
Task #1: Communicatio
Sub Task #3: Show the code

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server-side receiving the message

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response

Sub-Task

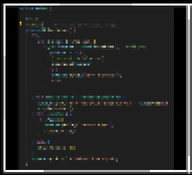
100%

Group: Communicatio
Task #1: Communicatio
Sub Task #4: Show the code

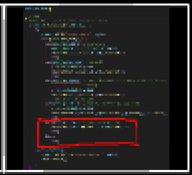
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Client receiving messages from the Server-side



Client receiving messages from the Server-side and presenting them

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must*

Response
Prompt

Briefly explain the code/logic/flow involved

Response:

Since the client is running, it gets the user's input from si.nextLine() and will be checked if it's a command or not. The message will then be sent using the sendMessage method with the user's value. In that method, it sets the payload type to MESSAGE and keeps its value which is later sent to the server using the send() method.

Prompt

Briefly explain the code/logic/flow involved

Response:

The processPayload() receives that payload which will determine if it's a MESSAGE type. It further uses the sendMessage method if it's truly a MESSAGE payload type. The sendMessage() method does the relay message job to every client in the room and the client will be removed if it fails.

be present)

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

In the listenToServer() method, it'll read the objects in stream of inputs and will process the payload method. In the processPayload(), it will handle the case of MESSAGE for simple messages and will be formated and then display the message desired.

End of Task 1

Task

100%

Group: Communication
Task #2: Rooms
Weight: ~50%
Points: ~1.50

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

Columns: 4

Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #1: Show Clients can	Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #2: Show Clients can	Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #3: Show the Client	Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #4: Show the
Task Screenshots	Task Screenshots	Task Screenshots	Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client creates a room

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

Gallery Style: 2 Columns

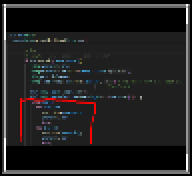
4 2 1

Shows client joining and leaving room

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

Gallery Style: 2 Columns

4 2 1


Client code for create/join room commands

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown (ucid/date must be present)

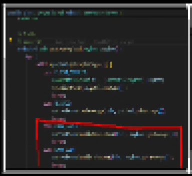
Task Response Prompt

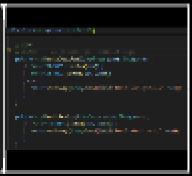
Briefly explain the code/logic/flow involved
Response:

With the processClientCommand method, it's able to check if the commnad that's passed is either a create/join room command. If so, it'll call the respective method which is sendCreateRoom or sendJoinroom and its value. The wasCommand will be set to true due to the true execution. The two methods have similar logic where it'll determine the payload type for either creating a room or joining a room and will se the room name and send that payload to the server.

Gallery Style: 2 Columns

4 2 1


ServerThread code for create/join handling


Room code handline create/join

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown (ucid/date must be present)

Task Response Prompt

Briefly explain the code/logic/flow involved
Response:

ServerThread: The handleCreateRoom and handleJoinRoom contains "this" which is reference to the serverThread that invokes the methods which gives access to clientId, etc. It then ses the payload class to get a string.

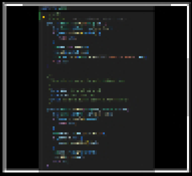
Room: For handleCreateRoom, it's going to communicate with the server instance and call the "createRoom" method in the server and going to take the room value. it'll then determine if it's true where the room is created or false if it was not. If the room was created, then "joinRoom" method will be called where it'll send the user to that new room based on the command value. If it fails, it'll simply state that the "room already exists" and can't create the same room. "handleJoinRoom" has a similar concept where it's responsbile for making that transition of moving the user to the room and seeing what room and by who. If it fails then it'll state "Room doesn't exist."

Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #5: Show the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Server code for handling the create/join

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

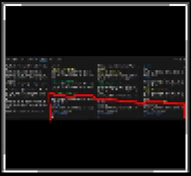
createRoom tries to create

Sub-Task 100% Group: Communicatio Task #2: Rooms Sub Task #6: Show that Client

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Messages are viewed by their respective room

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain why/how it works this way

Response:

a room and passes a name in which it'll be converted to lowercase to be consistent with other stuff. It'll then check the room's hashmap contains the key of nameCheck and will return false due to how a room already exits with that name. If not, then it'll create a new room with the name under that room. joinRoom passes the room name and the client who's wanting to join. If the room's hashmap doesn't contain the nameCheck then it'll return false. Meaning the room doesn't exist and can't join it. If not, then it'll get the client fromt the current room and be removed if it's not null. The "next" room is read and will add the client to that next room and return true.	Each client is set to be in the first room automatically created when they initally connect. Using the /create and /join command allows clients to have have these transitions towards multiple rooms. Now once the room is created, the client will be removed and transitioned to a new room created where now it will only read messages from the new room and nothing else. Due to how the client's have a thread, it will be cut with the previous room and only communicated in the new room. Any client that joins will be able to communicate with the clients in that room but not with the other rooms that are created unless they join.

End of Task 2

End of Group: Communication
Task Status: 2/2

Group

100%

Group: Disconnecting/Termination
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Disconnecting/Termination
Task #1: Disconnecting
Weight: ~100%
Points: ~3.00

COLLAPSE

Details:

Important: Code screenshots should be fairly concise (try to show only the sections of code relevant to the question)

Capturing all possible code (i.e., including a lot of irrelevant code) can lead to a reduced grade.

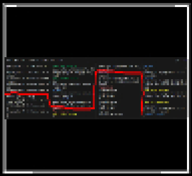
Columns: 4

Sub-Task 100% Group: Disconnecting/ Task #1: Disconnecting Sub Task #1: Show Clients

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Client "pepe" disconnected

Caption(s) (required) ✓

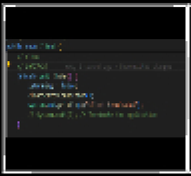
Caption Hint: *Describe/highlight what's being shown*

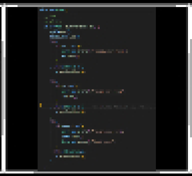
Sub-Task 100% Group: Disconnecting/ Task #1: Disconnecting Sub Task #2: Show the code

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Client disconnecting


Clients disconnecting

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

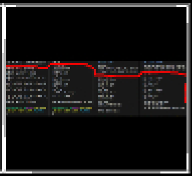
The close() and the closeServerConnection() do a process of cleaning up and associated resources. More specifically, the close() does a clean up towards the client connection where it will be stop thread

Sub-Task 100% Group: Disconnecting/ Task #1: Disconnecting Sub Task #3: Show the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Disconnected but later I ctrl-c which shows the later error.

Caption(s) (required) ✓

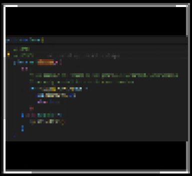
Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100% Group: Disconnecting/ Task #1: Disconnecting Sub Task #4: Show the Server

Task Screenshots

Gallery Style: 2 Columns

4 2 1


Server code handling termination

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (ucid/date must be present)*

Task Response Prompt

Briefly explain the code/logic/flow involved

Response:

The shutdown method will disconnect all the clients from the room created and rooms will also be removed.

by setting the client running status to false. It will call the closeServerConnection method which will close the server conneciton. As mentioned before, the closeServerConnection method does a clean up towards the server where it will reset the client data, close the output stream if the output isn't null. and close input stream if input isn't null. Then it will call the server class to use the close method to cloos the socket connection.

End of Task 1

End of Group: Disconnecting/Termination
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 this branch
Weight: ~33%
Points: ~0.33

COLLAPSE

Task URLs

URL #1
https://github.com/ElLopez21/el286-IT114-005/pull/9

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

End of Task 1

Task

Group: Misc

100%

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

COLLAPSE

Details:

Few related sentences about the Project/sockets topics

Task Response Prompt

Response:

Some issues I had along the way was getting the snippets of code such as for the server/rooms part and the parts where it you'd have to dig more deep. Each description of what we needed to find did give us the path of of kind of side the code needed to be retrieved but it took quite a lot of comprehension to get the necessary code. There were times where I would explain the code and the methods that are used I would skip them knowing that it's part of its process but forget to explain. I had to go back a lot of times to get that corrected. Also something minor that I forgot to do was to add my ucid and date and had to re-take some screenshots.

End of Task 2

Task

100%

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

COLLAPSE

Details:

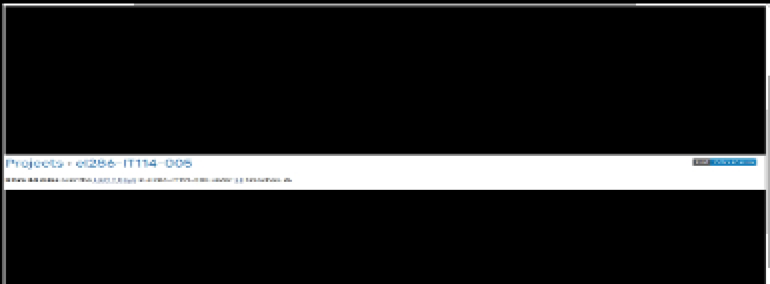
Grab a snippet showing the approximate time involved that clearly shows your repository.

The duration isn't considered for grading, but there should be some time involved.

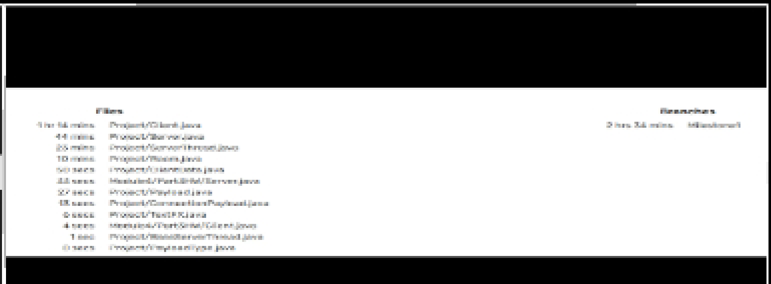
Task Screenshots

Gallery Style: 2 Columns

4 2 1


Projects • et286-IT114-006

[illegible]


Files

1 hr 14 mins	Project/Client.java
44 mins	Project/Server.java
25 mins	Project/ServerThread.java
10 mins	Project/Room.java
50 secs	Project/ClientUtils.java
44 secs	Module4/Part3HW/Server.java
27 secs	Project/Payload.java
18 secs	Project/ConnectionPayload.java
6 secs	Project/TextFX.java
4 secs	Module4/Part3HW/Client.java
1 sec	Project/RoomServerThread.java
0 secs	Project/PayloadType.java

Branches

2 hrs 34 mins	Milestone1

WakaTime Screenshot

WakaTime Screenshot

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment