Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-006-S2024/it114-project-milestone-1/grade/bm47

IT114-006-S2024 - [IT114] Project Milestone 1

Submissions:

Submission Selection

1 Submission [active] 3/21/2024 12:47:16 PM

Instructions

COLLAPSE

- Create a new branch called Milestone1
- At the root of your repository create a folder called Project if one doesn't exist yet
  - You will be updating this folder with new code as you do milestones
  - You won't be creating separate folders for milestones; milestones are just branches
- Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
- Copy in the latest Socket sample code from the most recent Socket Part example of the lessons
  - Recommended Part 5 (clients should be having names at this point and not ids)
  - https://github.com/MattToegel/IT114/tree/Module5/Module5
- Fix the package references at the top of each file (these are the only edits you should do at this point)
- Git add/commit the baseline and push it to github
- Create a pull request from Milestone1 to main (don't complete/merge it yet, just have it in open status)
- Ensure the sample is working and fill in the below deliverables
  - Note: The client commands likely are different in part 5 with the /name and /connect options instead of just "connect"
- Generate the worksheet output file once done and add it to your local repository
- Git add/commit/push all changes
- Complete the pull request merge from step 7
- Locally checkout main
- git pull origin main

Branch name: Milestone1

Tasks: 9 Points: 10.00

● **Start Up** (3 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Server and Client Initialization

Checklist *The checkboxes are for your own tracking

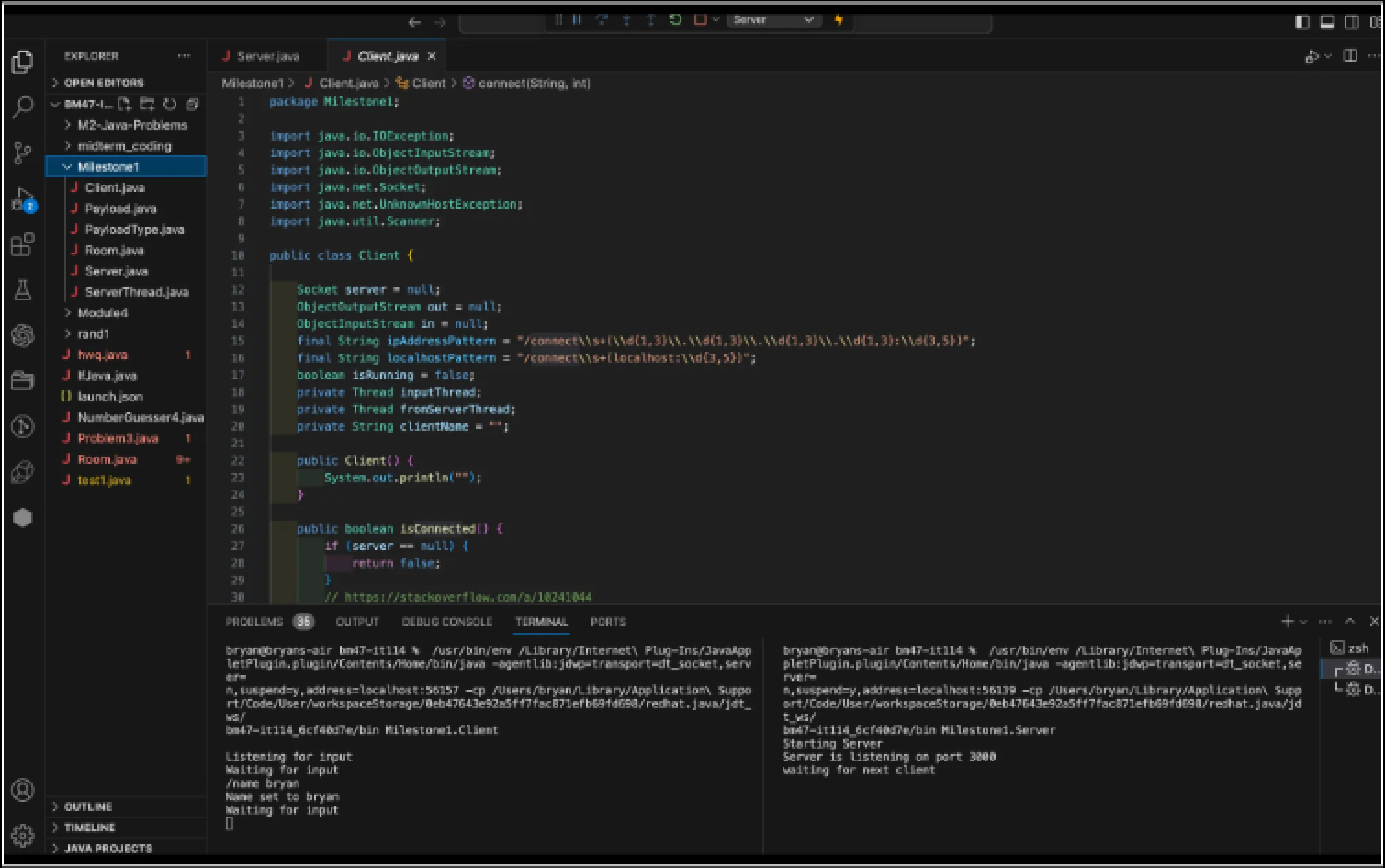
#	Points	Details
#1	1	Server should properly be listening to its port from the command line (note the related message)
#2	1	Clients should be successfully waiting for input
#3	1	Clients should have a name and successfully connected to the server (note related messages)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

1.



```java
package Milestone1;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class Client {

    Socket server = null;
    ObjectOutputStream out = null;
    ObjectInputStream in = null;
    final String ipAddressPattern = "/connect\\s+(\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}:\\d{3,5})";
    final String localhostPattern = "/connect\\s+(localhost:\\d{3,5})";
    boolean isRunning = false;
    private Thread inputThread;
    private Thread fromServerThread;
    private String clientName = "";

    public Client() {
        System.out.println("");
    }

    public boolean isConnected() {
        if (server == null) {
            return false;
        }
        // https://stackoverflow.com/a/10241044
```

```
bryan@bryans-air bm47-it114 %  /usr/bin/env /Library/Internet\ Plug-Ins/JavaApp
letPlugin.plugin/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,serv
er=
n,suspend=y,address=localhost:56157 -cp /Users/bryan/Library/Application\ Suppo
rt/Code/User/workspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jdt_
ws/
bm47-it114_6cf40d7e/bin Milestone1.Client

Listening for input
Waiting for input
/name bryan
Name set to bryan
Waiting for input
```

```
bryan@bryans-air bm47-it114 %  /usr/bin/env /Library/Internet\ Plug-Ins/JavaAp
pletPlugin.plugin/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,se
rver=
n,suspend=y,address=localhost:56139 -cp /Users/bryan/Library/Application\ Supp
ort/Code/User/workspaceStorage/0eb47643e92a5ff7fac871efb69fd698/redhat.java/jd
t_ws/
bm47-it114_6cf40d7e/bin Milestone1.Server
Starting Server
Server is listening on port 3000
waiting for next client
```

Server is listening on port 3000. Client is waiting for input. Client has a name.

Checklist Items (0)

Task #2 - Points: 1

Text: Explain the connection process

Details:

Note the various steps from the beginning to when the client is fully connected and able to communicate in the room.

Emphasize the code flow and the sockets usage.

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how the server-side of the connection works
#2	1	Mention how the client-side of the connection works
#3	1	Describe the socket steps until the server is waiting for messages from the client

Response:

The server-side of the connection works by listening to any incoming connection/traffic on a specific port.

The client then connects onto the given port and can input commands via the given port.
The information is exchanged by reading and writing to/from the socket on the given port.

Communication (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Add screenshot(s) showing evidence related to the checklist

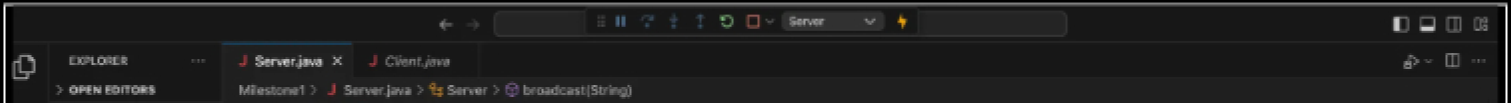
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	At least two clients connected to the server
#2	1	Client can send messages to the server
#3	1	Server sends the message to all clients in the same room
#4	1	Messages clearly show who the message is from (i.e., client name is clearly with the message)
#5	2	Demonstrate clients in two different rooms can't send/receive messages to each other (clearly show the clients are in different rooms via the commands demonstrated in the lessons
#6	1	Clearly caption each image regarding what is being shown

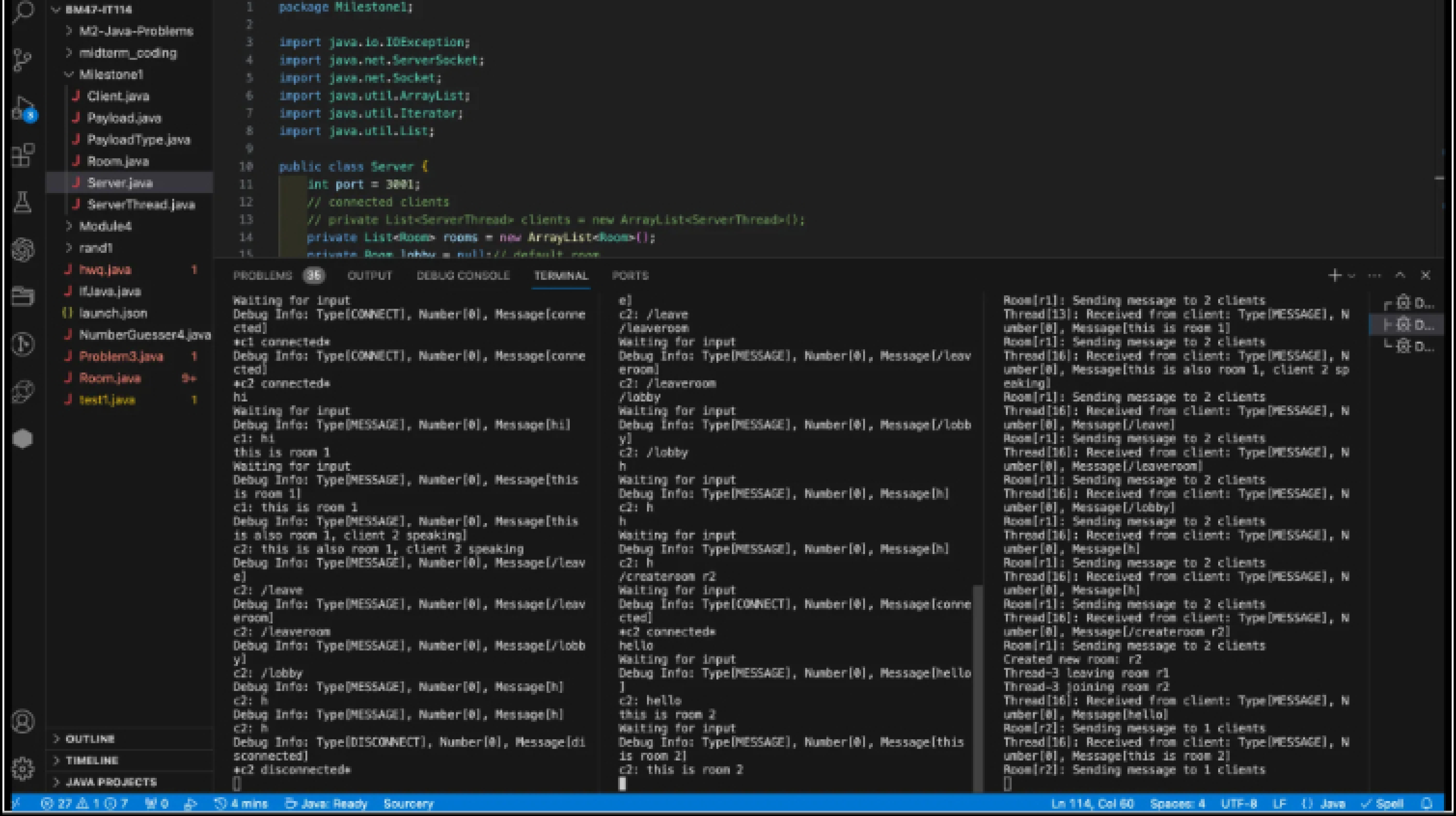
Task Screenshots:

Gallery Style: Large View

Small Medium Large



Milestone1 > Server.java > Server > broadcast(String)



```java
package Milestone1;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Server {
    int port = 3001;
    // connected clients
    // private List<ServerThread> clients = new ArrayList<ServerThread>();
    private List<Room> rooms = new ArrayList<Room>();
    private Room lobby = null;// default room
```

```
Waiting for input
Debug Info: Type[CONNECT], Number[0], Message[conne
cted]
*c1 connected*
Debug Info: Type[CONNECT], Number[0], Message[conne
cted]
*c2 connected*
hi
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[hi]
c1: hi
this is room 1
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[this
is room 1]
c1: this is room 1
Debug Info: Type[MESSAGE], Number[0], Message[this
is also room 1, client 2 speaking]
c2: this is also room 1, client 2 speaking
Debug Info: Type[MESSAGE], Number[0], Message[/leav
e]
c2: /leave
Debug Info: Type[MESSAGE], Number[0], Message[/leav
eroom]
c2: /leaveroom
Debug Info: Type[MESSAGE], Number[0], Message[/lobb
y]
c2: /lobby
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
Debug Info: Type[DISCONNECT], Number[0], Message[di
sconnected]
*c2 disconnected*
```

```
e]
c2: /leave
/leaveroom
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[/leav
eroom]
c2: /leaveroom
/lobby
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[/lobb
y]
c2: /lobby
h
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
h
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
/createroom r2
Waiting for input
Debug Info: Type[CONNECT], Number[0], Message[conne
cted]
*c2 connected*
hello
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[hello
]
c2: hello
this is room 2
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[this
is room 2]
c2: this is room 2
```

```
Room[r1]: Sending message to 2 clients
Thread[13]: Received from client: Type[MESSAGE], N
umber[0], Message[this is room 1]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[this is also room 1, client 2 sp
eaking]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[/leave]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[/leaveroom]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[/lobby]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[h]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[h]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[/createroom r2]
Room[r1]: Sending message to 2 clients
Created new room: r2
Thread-3 leaving room r1
Thread-3 joining room r2
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[hello]
Room[r2]: Sending message to 1 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[this is room 2]
Room[r2]: Sending message to 1 clients
```

The code shows first, client 1 (c1) and client 2 (c2) connected to the lobby, and then c1 creates room 1 (r1). Clients 1 and 2 join the room, and both clients can see the messages shown in the server console. After speaking, client 2 (c2) creates room 2 (r2) and speaks, and c1 can no longer see the message, as they are in a different room than c2. The messages also clearly depict the username in the room as asked in the checklist. All checklist items are met.

Checklist Items (0)

●

^COLLAPSE ^

Task #2 - Points: 1

Text: Explain the communication process

Details:

How are messages entered from the client side and how do they propagate to other clients?

Note all the steps involved and use specific terminology from the code.
Don't just translate the code line-by-line to plain English, keep it concise.

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention the client-side (sending)
#2	1	Mention the ServerThread's involvement
#3	1	Mention the Room's perspective
#4	1	Mention the client-side (receiving)

Response:

The client creates/connects to the room. When messages are sent, only clients inside of this room are able to see messages. It is written in the code that if the client is not in the room, the message is not seen.

The serverthread manages the rooms/messages between the rooms and ensures that multiple users can see each others messages if they are in the same room, and cannot see messages if they are not in the same room.
The room serves as a "channel" where only connected clients can see messages being sent between one another. Each client connected to the server is shown the message. If they are not in the server, the message is not sent to them.

The client presence in each room is checked in the code, if they are not present in the room, the message is not sent to them and they cannot see it.

● **Disconnecting/Termination** (3 pts.)

COLLAPSE

● COLLAPSE

Task #1 - Points: 1

Text: Add screenshot(s) showing evidence related to the checklist

Checklist

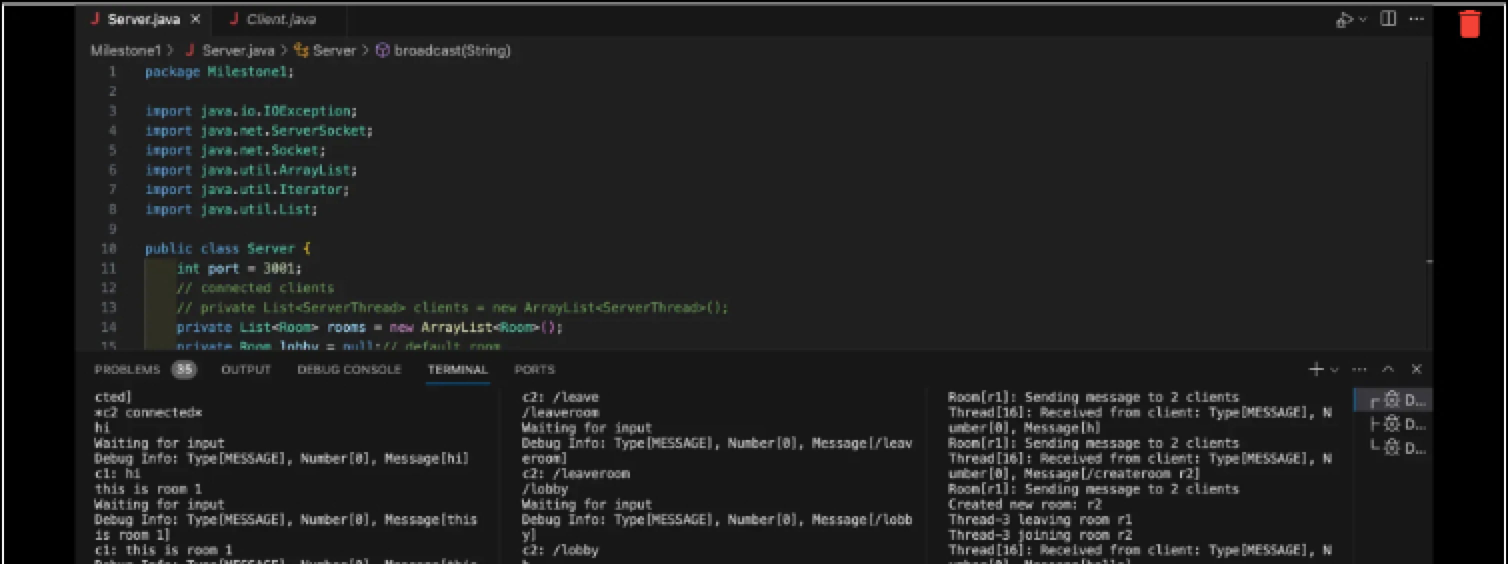
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show a client disconnecting from the server; Server should still be running without issue (it's ok if an exception message shows as it's part of the lesson code, the server just shouldn't terminate)
#2	1	Show the server terminating; Clients should be disconnected but still running and able to reconnect when the server is back online (demonstrate this)
#3	1	For each scenario, disconnected messages should be shown to the clients (should show a different person disconnected and should show the specific client disconnected)
#4	1	Clearly caption each image regarding what is being shown

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Server.java Client.java

Milestone1 > Server.java > Server > broadcast(String)

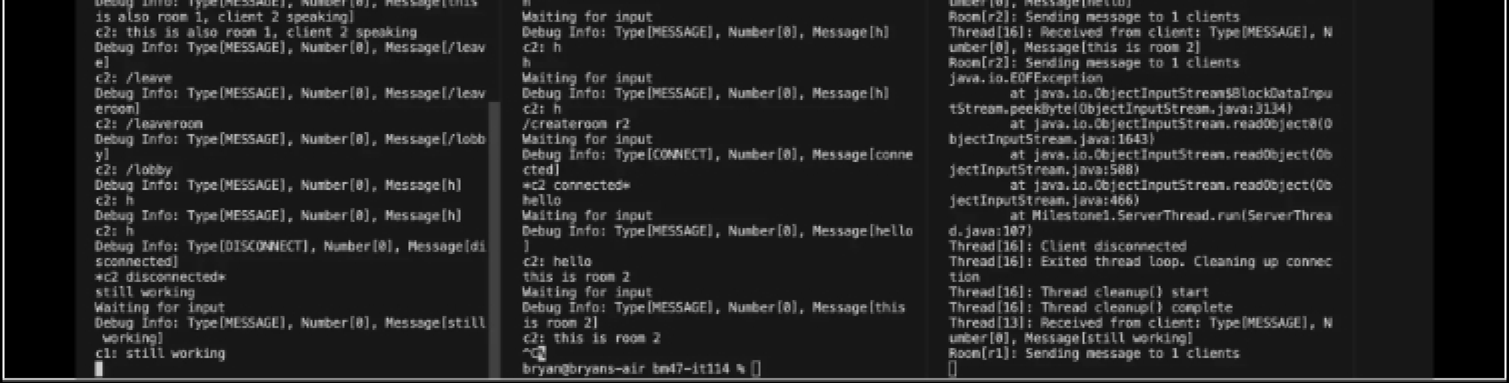
```
package Milestone1;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Server {
    int port = 3001;
    // connected clients
    // private List<ServerThread> clients = new ArrayList<ServerThread>();
    private List<Room> rooms = new ArrayList<Room>();
    private Room lobby = null;// default room
```

PROBLEMS 35 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
cted]
*c2 connected*
hi
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[hi]
c1: hi
this is room 1
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[this
is room 1]
c1: this is room 1
```

```
c2: /leave
/leaveroom
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[/leav
eroom]
c2: /leaveroom
/lobby
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[/lobb
y]
c2: /lobby
```

```
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[h]
Room[r1]: Sending message to 2 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[/createroom r2]
Room[r1]: Sending message to 2 clients
Created new room: r2
Thread-3 leaving room r1
Thread-3 joining room r2
Thread[16]: Received from client: Type[MESSAGE], N
```



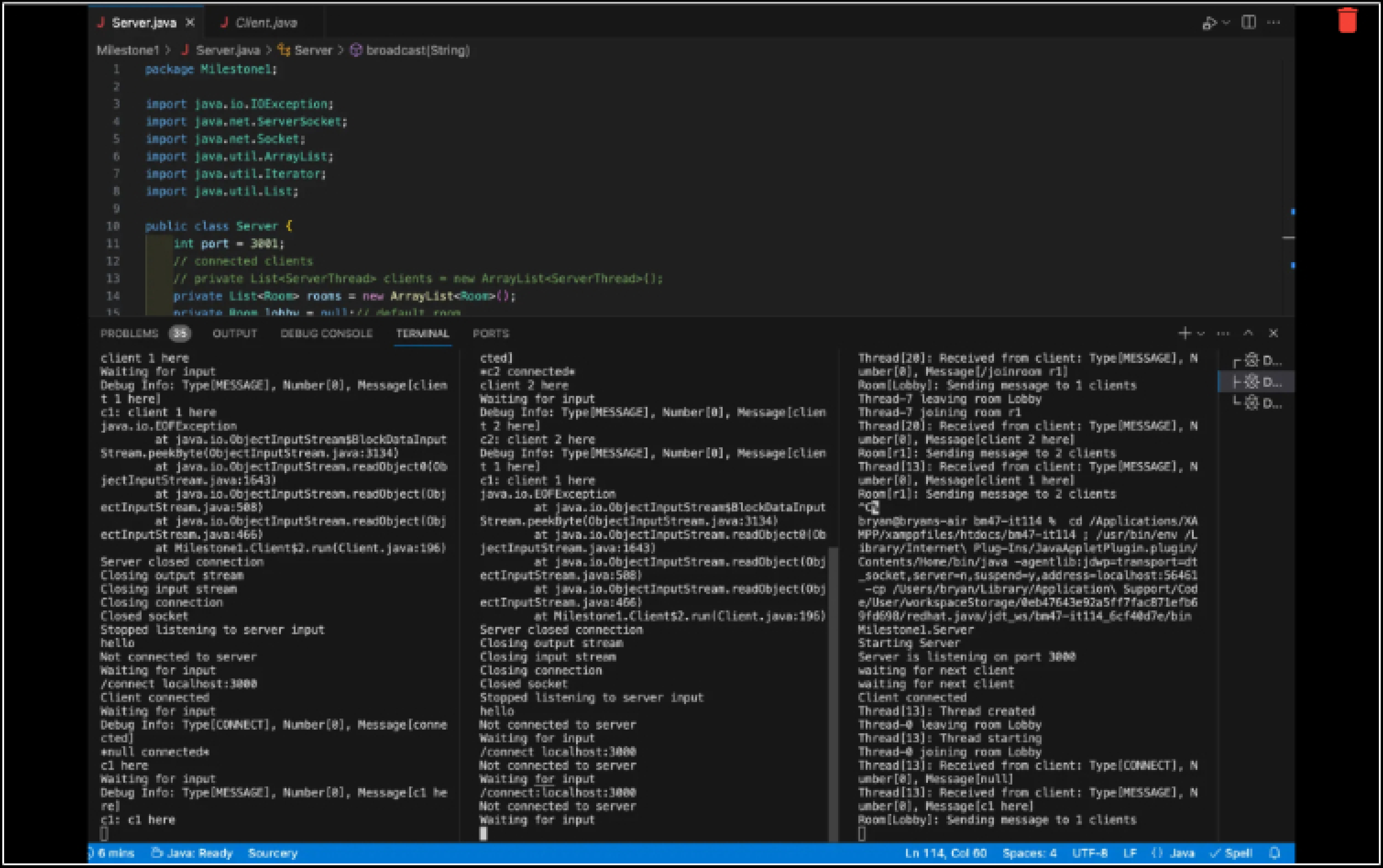
```
Debug Info: Type[MESSAGE], Number[0], Message[this
is also room 1, client 2 speaking]
c2: this is also room 1, client 2 speaking
Debug Info: Type[MESSAGE], Number[0], Message[/leav
e]
c2: /leave
Debug Info: Type[MESSAGE], Number[0], Message[/leav
eroom]
c2: /leaveroom
Debug Info: Type[MESSAGE], Number[0], Message[/lobb
y]
c2: /lobby
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
Debug Info: Type[DISCONNECT], Number[0], Message[di
sconnected]
*c2 disconnected*
still working
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[still
 working]
c1: still working
```

```
h
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
h
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[h]
c2: h
/createroom r2
Waiting for input
Debug Info: Type[CONNECT], Number[0], Message[conne
cted]
*c2 connected*
hello
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[hello
]
c2: hello
this is room 2
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[this
is room 2]
c2: this is room 2
^C
bryan@bryans-air bm47-it114 %
```

```
umber[0], Message[hello]
Room[r2]: Sending message to 1 clients
Thread[16]: Received from client: Type[MESSAGE], N
umber[0], Message[this is room 2]
Room[r2]: Sending message to 1 clients
java.io.EOFException
        at java.io.ObjectInputStream$BlockDataInpu
tStream.peekByte(ObjectInputStream.java:3134)
        at java.io.ObjectInputStream.readObject0(O
bjectInputStream.java:1643)
        at java.io.ObjectInputStream.readObject(Ob
jectInputStream.java:508)
        at java.io.ObjectInputStream.readObject(Ob
jectInputStream.java:466)
        at Milestone1.ServerThread.run(ServerThrea
d.java:107)
Thread[16]: Client disconnected
Thread[16]: Exited thread loop. Cleaning up connec
tion
Thread[16]: Thread cleanup() start
Thread[16]: Thread cleanup() complete
Thread[13]: Received from client: Type[MESSAGE], N
umber[0], Message[still working]
Room[r1]: Sending message to 1 clients
```

Image 1 shows client 2 (c2) disconnecting from the server. I then sent a message as client 1 (c1) saying that the server/client connected and the room is still working properly.

Checklist Items (0)



```java
package Milestone1;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Server {
    int port = 3001;
    // connected clients
    // private List<ServerThread> clients = new ArrayList<ServerThread>();
    private List<Room> rooms = new ArrayList<Room>();
    private Room lobby = null;// default room
```

```
client 1 here
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[clien
t 1 here]
c1: client 1 here
java.io.EOFException
        at java.io.ObjectInputStream$BlockDataInput
Stream.peekByte(ObjectInputStream.java:3134)
        at java.io.ObjectInputStream.readObject0(Ob
jectInputStream.java:1643)
        at java.io.ObjectInputStream.readObject(Obj
ectInputStream.java:508)
        at java.io.ObjectInputStream.readObject(Obj
ectInputStream.java:466)
        at Milestone1.Client$2.run(Client.java:196)
Server closed connection
Closing output stream
Closing input stream
Closing connection
Closed socket
Stopped listening to server input
hello
Not connected to server
Waiting for input
/connect localhost:3000
Client connected
Waiting for input
Debug Info: Type[CONNECT], Number[0], Message[conne
cted]
*null connected*
c1 here
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[c1 he
re]
c1: c1 here
```

```
cted]
*c2 connected*
client 2 here
Waiting for input
Debug Info: Type[MESSAGE], Number[0], Message[clien
t 2 here]
c2: client 2 here
Debug Info: Type[MESSAGE], Number[0], Message[clien
t 1 here]
c1: client 1 here
java.io.EOFException
        at java.io.ObjectInputStream$BlockDataInput
Stream.peekByte(ObjectInputStream.java:3134)
        at java.io.ObjectInputStream.readObject0(Ob
jectInputStream.java:1643)
        at java.io.ObjectInputStream.readObject(Obj
ectInputStream.java:508)
        at java.io.ObjectInputStream.readObject(Obj
ectInputStream.java:466)
        at Milestone1.Client$2.run(Client.java:196)
Server closed connection
Closing output stream
Closing input stream
Closing connection
Closed socket
Stopped listening to server input
hello
Not connected to server
Waiting for input
/connect localhost:3000
Not connected to server
Waiting for input
/connect:localhost:3000
Not connected to server
Waiting for input
```

```
Thread[20]: Received from client: Type[MESSAGE], N
umber[0], Message[/joinroom r1]
Room[Lobby]: Sending message to 1 clients
Thread-7 leaving room Lobby
Thread-7 joining room r1
Thread[20]: Received from client: Type[MESSAGE], N
umber[0], Message[client 2 here]
Room[r1]: Sending message to 2 clients
Thread[13]: Received from client: Type[MESSAGE], N
umber[0], Message[client 1 here]
Room[r1]: Sending message to 2 clients
^C
bryan@bryans-air bm47-it114 %  cd /Applications/XA
MPP/xamppfiles/htdocs/bm47-it114 ; /usr/bin/env /L
ibrary/Internet\ Plug-Ins/JavaAppletPlugin.plugin/
Contents/Home/bin/java -agentlib:jdwp=transport=dt
_socket,server=n,suspend=y,address=localhost:56461
 -cp /Users/bryan/Library/Application\ Support/Cod
e/User/workspaceStorage/0eb47643e92a5ff7fac871efb6
9fd698/redhat.java/jdt_ws/bm47-it114_6cf40d7e/bin
Milestone1.Server
Starting Server
Server is listening on port 3000
waiting for next client
waiting for next client
Client connected
Thread[13]: Thread created
Thread-0 leaving room Lobby
Thread[13]: Thread starting
Thread-0 joining room Lobby
Thread[13]: Received from client: Type[CONNECT], N
umber[0], Message[null]
Thread[13]: Received from client: Type[MESSAGE], N
umber[0], Message[c1 here]
Room[Lobby]: Sending message to 1 clients
```

Image 2 here shows the server shutting down using CTRL+C to terminate it. I then restart the server and joined the room with c1, and messages are now working properly again. There was an issue sending messages with c2 after starting the server as you can see, and I was not able to determine why, but c1 worked properly after starting the server again.

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Explain the various Disconnect/termination scenarios

Details:

Include the various scenarios of how a disconnect can occur. There should be around 3 or so.

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how a client gets disconnected from a Socket perspective
#2	1	Mention how/why the client program doesn't crash when the server disconnects/terminates.
#3	1	Mention how the server doesn't crash from the client(s) disconnecting

Response:

Sockets close their connection using the close() method written in the code. It then stops sending/receiving information via the socket.
2) Clients do not know if a socket closes its connection, so that it why it does not crash. One way to see if the socket closed the connection is by sending information and it is determined by the response received/where the information goes.
3) Similar to clients, sockets also do not necessarily know that a client closes a connection unless it checks using one of the methods. Additionally, it would not crash because the server can operate independently whether or not there are multiple or 0 clients connected. Also, the server can check if a client has disconnected by sending bytes to the client and the status of the client is determined by the response.

● Misc (1 pt.)

COLLAPSE

● COLLAPSE

Task #1 - Points: 1

Text: Add the pull request link for this branch

URL #1

Missing URL

● COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Details:

Few related sentences about the Project/sockets topics

Response:

Missing Response

● 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

Missing Caption

End of Assignment