Practical-2

Aim:

The aim of this project is to design and implement a simple chat server and client application in Java that supports basic functionalities like broadcasting messages to all connected clients, sending direct messages between clients, listing active users, and handling client disconnections after inactivity.

Theory:

A chat server is a system designed to manage and facilitate communication between multiple clients. It involves two main components: the server, which listens for incoming connections and manages the communication between clients, and the clients, which connect to the server and send/receive messages.

The core components of this chat application include:

1. **Socket Programming:** Sockets provide the communication mechanism between two computers using TCP (Transmission Control Protocol). In this project, the ServerSocket class is used by the server to listen for incoming connections, and the Socket class is used by the client to establish a connection to the server.
2. **Multithreading:** To handle multiple clients simultaneously, the server uses multithreading. Each client is managed by a separate thread, ensuring that the server can process multiple requests concurrently without blocking other clients.
3. **Timeout Handling:** To manage inactive clients, a timeout mechanism is implemented. If a client remains inactive for a specified period, the server disconnects the client to free up resources.
4. **Message Broadcasting and Direct Messaging:** The server can broadcast messages to all clients or relay direct messages between specific clients based on their unique call signs.

Function & Modules Used:

1. **ChatServer Class:**

   - **ServerSocket:** Used to listen for incoming client connections on a specified port.
   - **ArrayList<ClientHandler> clients:** Maintains a list of all connected clients.
   - **HashMap<String, ClientHandler> clientMap:** Maps client call signs to their corresponding ClientHandler objects for easy retrieval during direct messaging.
   - **ExecutorService pool:** Manages a pool of threads to handle client connections and tasks efficiently.

   Methods:

   - run(): Starts the server and listens for incoming connections.
   - broadcast(String message): Sends a message to all connected clients.
   - shutServer(): Shuts down the server and releases resources.
   - log(String message): Logs server events with a timestamp.

2. **ClientHandler Class (Nested inside ChatServer):**

   - **Socket client:** Represents the connection between the server and a client.
   - **BufferedReader in, PrintWriter out:** Used for reading messages from and sending messages to the client.
   - **TimeoutHandler timeoutHandler:** Manages client inactivity by implementing a timeout mechanism.

   Methods:

   - run(): Handles client communication, including reading client messages and responding appropriately.
   - sendMessageToClient(String targetCallSign, String message): Sends a direct message to a specific client based on their call sign.

- shutClient(): Closes the client connection and releases resources.
- sendMessage(String message): Sends a message to the connected client.
- getCallSign(): Retrieves the client's call sign.

3. **ChatClient Class:**

   - **Socket clientSocket:** Establishes a connection to the server.
   - **PrintWriter out, BufferedReader in:** Used for sending messages to and receiving messages from the server.
   - **String callSign:** Represents the client's unique identifier in the chat system.
   - **Thread serverListenerThread:** Listens for messages from the server in a separate thread.

   Methods:

   - startConnection(String ip, int port): Establishes a connection to the server.
   - sendMessage(String msg): Sends a message to the server.
   - stopConnection(): Closes the connection to the server.
   - startServerListener(): Listens for incoming messages from the server and displays them to the client.
   - run(): Manages client interaction, including setting up the connection, handling user input, and managing the connection lifecycle.

Analysis:

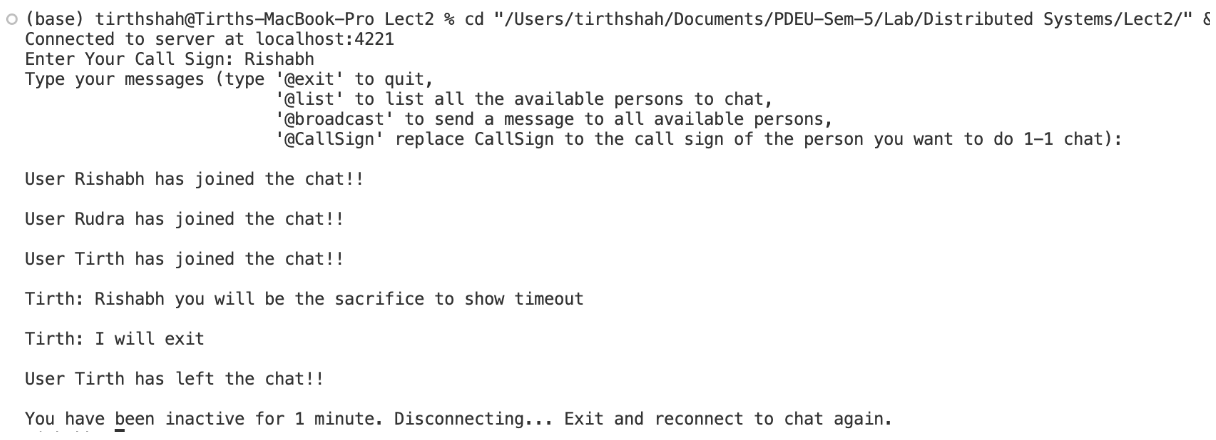
The chat server and client application were tested under various conditions to evaluate its performance and functionality. The following key aspects were analyzed:

1. **Concurrency Handling:** The server successfully managed multiple client connections simultaneously without any noticeable delay or resource contention, demonstrating the effectiveness of the multithreading approach.
2. **Timeout Mechanism:** The timeout handler effectively disconnected inactive clients after 1 minute of inactivity, ensuring that server resources were not wasted on idle connections.

3. **Message Broadcasting and Direct Messaging:** Both broadcasting and direct messaging functionalities worked as expected. Clients could easily send messages to all users or target specific users using the appropriate commands.
4. **Error Handling:** The server handled various exceptions gracefully, including client disconnections, network failures, and invalid commands, maintaining overall system stability.

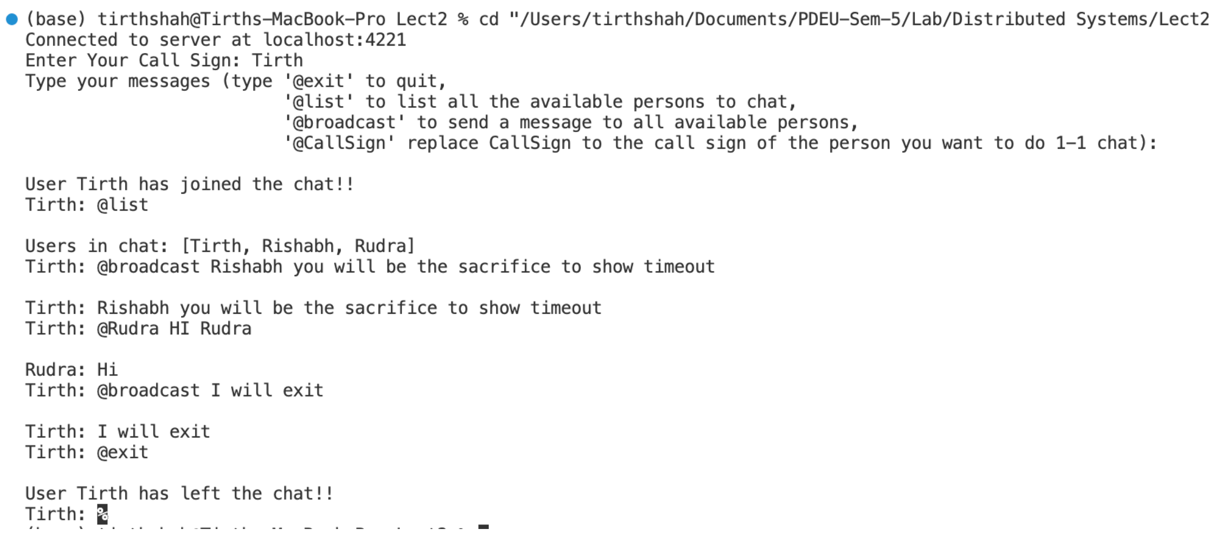
Test Case and Output:

1. Rishabh was demonstrating how it will get disconnected from the chat after inactivity.

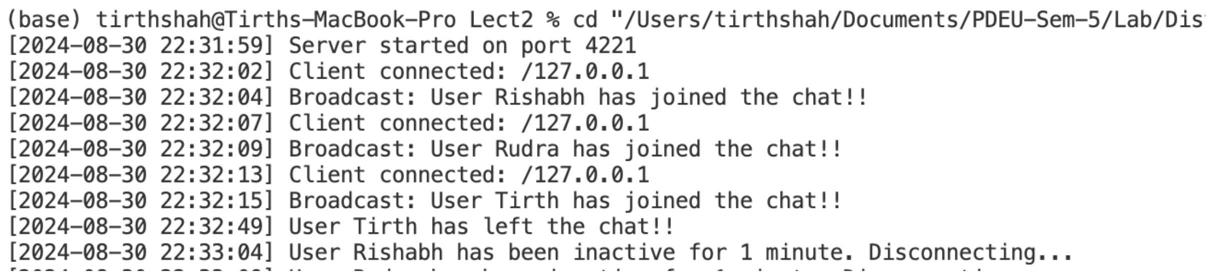


```
○ (base) tirthshah@Tirths-MacBook-Pro Lect2 % cd "/Users/tirthshah/Documents/PDEU-Sem-5/Lab/Distributed Systems/Lect2/" &
  Connected to server at localhost:4221
  Enter Your Call Sign: Rishabh
  Type your messages (type '@exit' to quit,
                           '@list' to list all the available persons to chat,
                           '@broadcast' to send a message to all available persons,
                           '@CallSign' replace CallSign to the call sign of the person you want to do 1-1 chat):

  User Rishabh has joined the chat!!

  User Rudra has joined the chat!!

  User Tirth has joined the chat!!

  Tirth: Rishabh you will be the sacrifice to show timeout

  Tirth: I will exit

  User Tirth has left the chat!!

  You have been inactive for 1 minute. Disconnecting... Exit and reconnect to chat again.
```

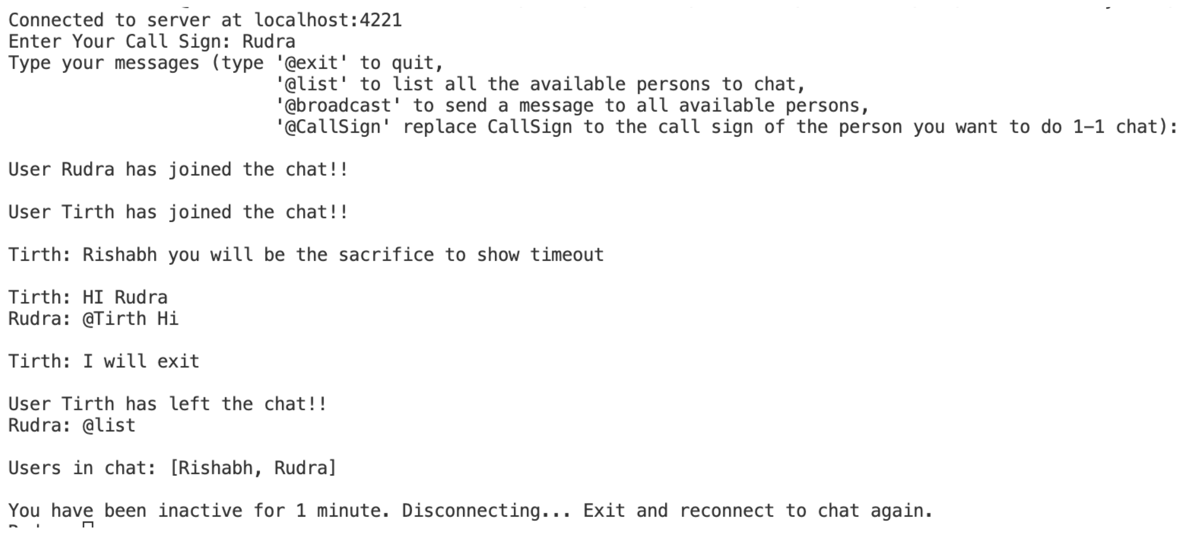
2. Tirth will show broadcast and list facilities and will show exit facility showing that it will not disturb server. It will also show how he chatted with Rudra



```
● (base) tirthshah@Tirths-MacBook-Pro Lect2 % cd "/Users/tirthshah/Documents/PDEU-Sem-5/Lab/Distributed Systems/Lect2
  Connected to server at localhost:4221
  Enter Your Call Sign: Tirth
  Type your messages (type '@exit' to quit,
                           '@list' to list all the available persons to chat,
                           '@broadcast' to send a message to all available persons,
                           '@CallSign' replace CallSign to the call sign of the person you want to do 1-1 chat):

  User Tirth has joined the chat!!
  Tirth: @list

  Users in chat: [Tirth, Rishabh, Rudra]
  Tirth: @broadcast Rishabh you will be the sacrifice to show timeout

  Tirth: Rishabh you will be the sacrifice to show timeout
  Tirth: @Rudra HI Rudra

  Rudra: Hi
  Tirth: @broadcast I will exit

  Tirth: I will exit
  Tirth: @exit

  User Tirth has left the chat!!
  Tirth: %
```



```
(base) tirthshah@Tirths-MacBook-Pro Lect2 % cd "/Users/tirthshah/Documents/PDEU-Sem-5/Lab/Dis
[2024-08-30 22:31:59] Server started on port 4221
[2024-08-30 22:32:02] Client connected: /127.0.0.1
[2024-08-30 22:32:04] Broadcast: User Rishabh has joined the chat!!
[2024-08-30 22:32:07] Client connected: /127.0.0.1
[2024-08-30 22:32:09] Broadcast: User Rudra has joined the chat!!
[2024-08-30 22:32:13] Client connected: /127.0.0.1
[2024-08-30 22:32:15] Broadcast: User Tirth has joined the chat!!
[2024-08-30 22:32:49] User Tirth has left the chat!!
[2024-08-30 22:33:04] User Rishabh has been inactive for 1 minute. Disconnecting...
```



```
Connected to server at localhost:4221
Enter Your Call Sign: Rudra
Type your messages (type '@exit' to quit,
                         '@list' to list all the available persons to chat,
                         '@broadcast' to send a message to all available persons,
                         '@CallSign' replace CallSign to the call sign of the person you want to do 1-1 chat):

User Rudra has joined the chat!!

User Tirth has joined the chat!!

Tirth: Rishabh you will be the sacrifice to show timeout

Tirth: HI Rudra
Rudra: @Tirth Hi

Tirth: I will exit

User Tirth has left the chat!!
Rudra: @list

Users in chat: [Rishabh, Rudra]

You have been inactive for 1 minute. Disconnecting... Exit and reconnect to chat again.
```

Conclusion:

The project demonstrates the practical application of several key concepts in Java networking and concurrency.

The chat server is capable of handling real-time communication between multiple clients, offering both broadcast and private messaging functionalities. The inclusion of a timeout mechanism for inactive clients enhances the server's resource management and ensures that it remains responsive and available for active users.

This implementation could be further extended to include features such as secure communication, adding support for sending multimedia files, persistent user sessions, or integration with a database for storing chat history.