Client-Server Chat Application using TCP Sockets

1 Introduction

This document explains the structure and functionality of a simple client-server chat application using TCP sockets in Java. The system consists of three main components:

- **Server:** Accepts connections and manages clients.
- **ClientHandler:** Handles communication with a single client.
- **Client:** Connects to the server and sends/receives messages.

2 Server Class (`Server.java`)

This class creates a server that listens for connections and manages communication.

2.1 Key Objects and Methods

- `ServerSocket serverSocket = new ServerSocket(port);`
  - Creates a **server socket** that listens on a specified port (e.g., 12345).
  - Accepts incoming client connections.
- `Socket socket = serverSocket.accept();`
  - Blocks execution until a client connects.
  - Returns a `Socket` object representing the connection.
- `new ClientHandler(socket).start();`
  - Creates a new thread to handle each connected client.
  - `ClientHandler` class manages **reading/writing messages**.

3 ClientHandler Class (`ClientHandler.java`)

Each client runs on a separate thread.

3.1 Key Objects and Methods

- `BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));`
  - Reads data from the client's **input stream** (messages sent by the client).
- `PrintWriter out = new PrintWriter(socket.getOutputStream(), true);`
  - Sends data to the client's **output stream**.
  - The `true` argument enables **auto-flush** (sends data immediately).
- `while ((message = in.readLine()) != null) { ... }`
  - Reads messages from the client until the connection is closed.
- `out.println("Server: " + message.toUpperCase());`
  - Sends a **response** back to the client.
- `socket.close();`
  - Closes the connection when the client disconnects.

4 Client Class (`Client.java`)

The client connects to the server and sends messages.

4.1 Key Objects and Methods

- `Socket socket = new Socket(serverAddress, port);`
  - Creates a connection to the **server IP and port**.
- `BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));`
  - Reads **user input** from the keyboard.
- `BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));`
  - Reads **server responses**.
- `PrintWriter out = new PrintWriter(socket.getOutputStream(), true);`
  - Sends messages to the server.

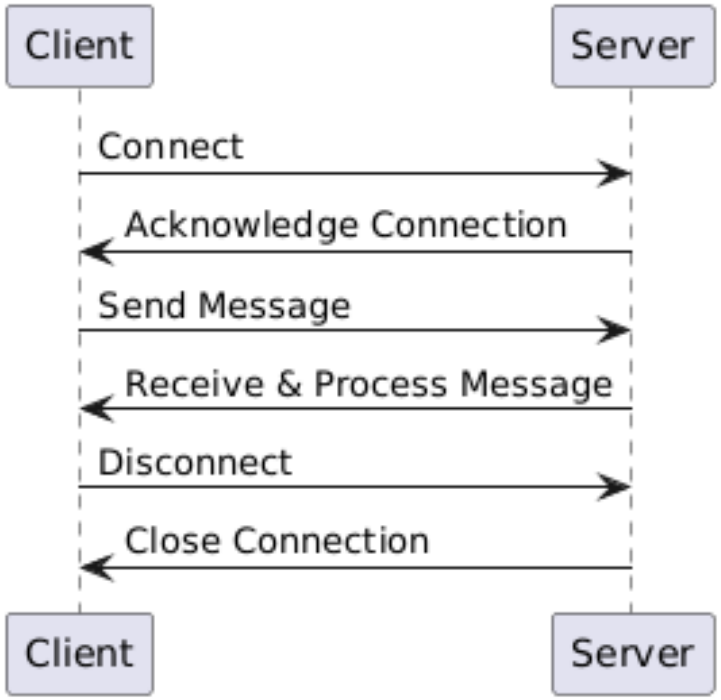
- `while (true) { message = userInput.readLine(); out.println(message); }`
  - Takes **user input** and **sends it to the server** continuously.
  - Reads the **server response** and prints it.

5 Summary

The table below summarizes the key responsibilities and objects for each class:

Class	Main Responsibility	Key Objects
Server	Accepts connections & starts client threads	`ServerSocket`, `Socket`
ClientHandler	Handles communication for one client	`BufferedReader`, `PrintWriter`, `Thread`
Client	Connects to server & sends messages	`Socket`, `BufferedReader`, `PrintWriter`

Table 1: Client-Server Chat Application Components



figureSocket ChatApp TCP-One Client Chat System Flow