Report: Client-Server Chat Application in Java

Project Overview

This project implements a simple client-server chat application in Java. The server runs on localhost at port 5000, and the client can connect to it, send messages, and receive server responses in real-time. The application uses **Java Sockets** and **multithreading** to handle communication between multiple clients and the server.

Technologies Used

- **Java** (JDK 8 or later)
- **Socket Programming** (Client and Server communication over TCP/IP)
- **Multithreading** (Handling multiple clients concurrently)

Project Structure

1. **CreateServer.java**
    - This is the server application. It listens for client connections and allows two-way communication with the client. The server accepts messages from clients and allows the server to send responses back.
2. **ClientChat.java**
    - This is the client application. It connects to the server on localhost (port 5000), sends messages, and listens for responses from the server.

How It Works

Server Side (CreateServer.java)

1. The server listens for client connections on port 5000.
2. When a client connects, the server establishes input and output streams for communication (BufferedReader for reading and PrintWriter for writing).
3. A new thread is started to handle incoming messages from the client. This thread listens for messages sent by the client and prints them to the console.
4. The main thread listens for input from the server (user input in the terminal). When the user types a message, it is sent to the connected client.
5. The server continues this process until the server input reads "exit", at which point the server closes the client connection and shuts down.

Client Side (ClientChat.java)

1. The client connects to the server on localhost (IP address 127.0.0.1) and port 5000.
2. Once connected, the client starts listening for incoming messages from the server (in a separate thread).
3. The client also allows the user to input messages, which are sent to the server.
4. The client receives messages from the server and displays them in the console.
5. The client continues this process until it is disconnected, or the user terminates the program.

How to Run the Application

1. **Compile the Code:**
    - Open your terminal or command prompt and navigate to the directory where the Java files are saved.
    - Compile the server and client programs using the following commands:

        javac CreateServer.java

        javac ClientChat.java
2. **Run the Server:**
    - In the terminal, run the server program:

        java CreateServer
    - The server will start and wait for clients to connect.
3. **Run the Client:**
    - In a new terminal window, run the client program:

        java ClientChat
    - The client will connect to the server on localhost (IP: 127.0.0.1) and port 5000.
4. **Start Chatting:**
    - The client and server can now exchange messages. The server can send messages to the client, and the client can send messages to the server.
    - The server can type messages, and the client will display them. The client can also type messages, which will be displayed on the server side.

5. **Exit the Application:**
    - To close the application, the server needs to type exit in the terminal, which will stop the server and close the client connection.

Key Features

- **Multithreading:** Handles multiple clients in separate threads for concurrent communication.
- **Real-Time Messaging:** Messages are transmitted between the client and server in real time.
- **Graceful Termination:** The server can gracefully close the connection with the client by typing exit.

Improvements for Future Versions

- **Multiple Clients Support:** The server can be enhanced to handle multiple clients concurrently.
- **GUI:** The chat application could be developed with a graphical user interface (GUI) to make it more user-friendly.
- **Message History:** Implementing message history so that both the client and server can keep track of past messages.

Conclusion

This simple chat application demonstrates basic client-server communication in Java, focusing on socket programming and multithreading. It's an excellent foundation for building more complex applications that involve network communication.