**README**

**Make a new project with WindowBuilder to use JFrame**

**Run Server.java**

**The Chat Server is running..**

**Run Client.java**

**You can open multiple clients by running Client.java files multiple times.**

**Multiple Client – Chat Application**

***INTRODUCTION***

The stream communication protocol is known as TCP (transfer control protocol). TCP is a connection-oriented protocol. It works as described in this document. In order to communicate over the TCP protocol, a connection must first be established between two sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once the two sockets are connected, they can be used to transmit and/or to receive data. When we say "two sockets are connected" we mean the fact that the server accepted a connection. As it was explained above the server creates a new local socket for the new connection. The process of the new local socket creation, however, is transparent for the client.

The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol. No connection is established before sending the data. The data are sent in a packet called datagram. The datagram is sent like a request for establishing a connection. However, the datagram contains not only the addresses, it contains the user data also. Once it arrives to the destination the user data are read by the remote application and no connection is established. This protocol requires that each time a datagram is sent, the local socket and the remote socket addresses must also be sent in the datagram. These addresses are sent in each datagram.

The **java.net** package in the Java development environment provides the class **DatagramSocket** for programming datagram communications.

A chat application consists of a chat server and a chat client. The server accepts connections from the clients and delivers all messages from each client to other clients. This is a tool to communicate with other people over Internet in real time.

The client is implemented using two threads - one thread to interact with the server and the other with the standard input. Two threads are needed because a client must communicate with the server and, simultaneously, it must be ready to read messages from the standard input to be sent to the server.  
The server is implemented using threads also. It uses a separate thread for each connection. It spawns a new client thread every time a new connection from a client is accepted. This simplifies a lot the design of the server. Multi-threading, however, creates synchronization issues. We will present two implementations of the chat server. An implementation that focus on multi-threading without considering the synchronization issues will be presented first. Then we will focus on the synchronization issues that a multi-threaded implementation creates. Finally, an updated version of the multi-threaded chat server that fixes the synchronization issues is presented.

***IMPLEMENTATION***

Technology Used : Eclipse , Socket Programming

Language Used : Java

A multithreaded chat room server.  When a client connects the server requests a screen name by sending the client the text "SUBMITNAME", and keeps requesting a name until a unique one is received.  After a client submits a unique name, the server acknowledges with "NAMEACCEPTED".  Then all messages from that client will be broadcast to all other clients that have submitted a unique screen name.  The broadcast messages are prefixed with "MESSAGE ".

Because this is just a teaching example to illustrate a simple chat server, there are a few features that have been left out. Two are very useful and belong in production code:

1. The protocol should be enhanced so that the client can send clean disconnect messages to the server.

2. The server should do some logging.The ChatClient.java simply creates  socket  connection with the specified address on port  9001.Once a connection is established, two threads are creating. One for reading from the socket and other for writing to socket. Once the server disconnects the connection, the client exits itself.

***Graphical User Interface***

**1. Enter the IP Address of your local machine**

**Graphical user interface, application

Description automatically generated**

Graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

***Conclusion***

Java sockets API (Socket and ServerSocket classes) is a powerful and flexible interface for network programming of client/server applications.

On the other hand, Java threads is another powerful programming framework for client/server applications. Multi-threading simplifies the implementation of complex client/server applications.