Introduction

In this article I am demonstrating a chat application which can handle multiple users at the same time. It also supports file transfer.

It is entirely based on Java and consists of two parts: jMessenger (client application) and jServer (server application).

Features

1. Handles multiple users at the same time
2. Support for both public and private messages
3. User signup and login available
4. Support for file transfer

Using the code

Run the jar files *jMessenger.jar* and *jServer.jar* and do the following:

1. On jServer select "*data.xml*" as database file. This file contains usernames and passwords.
2. On jMessenger select "*History.xml*" as history file. This file is used to save chat history.
3. In many cases, if jMessenger cannot find the server then adjust firewall to give it network access.

Both applications are written in Netbeans and you can import source files in Netbeans to view and edit them.

Message structure

Each message in jMessenger has four fields:

- `type`: This can be set to `message`, `login`, `newuser`, etc.
- `sender`: The username of sender
- `content`: Actual content of the message
- `Recipient`: Username of recipient of the message

jServer

There are two main classes in **jServer** for handling connections and messages. On startup the `SocketServer`runs in a separate thread. The job of `SocketServer` is to wait for connections and for each connection start a new thread `ServerThread`. Once the connection is established, `ServerThread` will listen for any messages and hand it over to `SocketServer` to process. Also it will forward messages from other users to the connected user.

Copy Code

```
// In ServerThread read the incoming message and hand it to SocketServer

Message msg = (Message) streamIn.readObject();
server.handle(ID, msg);
.......


// In SocketServer process the messages based on their type

public synchronized void handle(int ID, Message msg){
         if(msg.type.equals("login")){
                  ....
         }
         else id(msg.type.equals("message")){
                  if(msg.recipient.equals("All")){ Announce("message", msg.sender,
msg.content); }
                  else{
                           // Find the thread of recipient and forward it to him
                  }
         }
.......
```

jMessenger

jMessenger first connects to the jServer, specified by its IP-address and port number. Arriving messages are then displayed on message board along with their senders.

When a user wants to send a file, first his request is sent via a message of type `upload_req`. The recipient then does the following:

1. The recipient side sends its reply in a message of type `upload_res`
2. If request is accepted then the recipient opens a new port
3. For positive reply, recipient's IP address and port number is sent back
4. The sender, on receiving positive reply connects to this socket and starts file upload

An advantage of this approach is that the clients can chat and transfer files at the same time. Unlike messages, files do not go through jServer.

```
// On recipient side, start a new thread for download

Download dwn = new Download(....);
Thread t = new Thread(dwn);
t.start();
send(new Message("upload_res", ui.username, dwn.port, msg.sender));
// Reply to sender with IP address and port number
.........

// On sender side, start a new thread for file upload

// Connect to the port specified in reply
Upload upl = new Upload(addr, port, ui.file, ui);
Thread t = new Thread(upl);
t.start();
```