

# **INSTANT MESSAGING PROGRAM**

di  
Camilla Franceschini (282280)  
e  
Mattia Greci (286789)

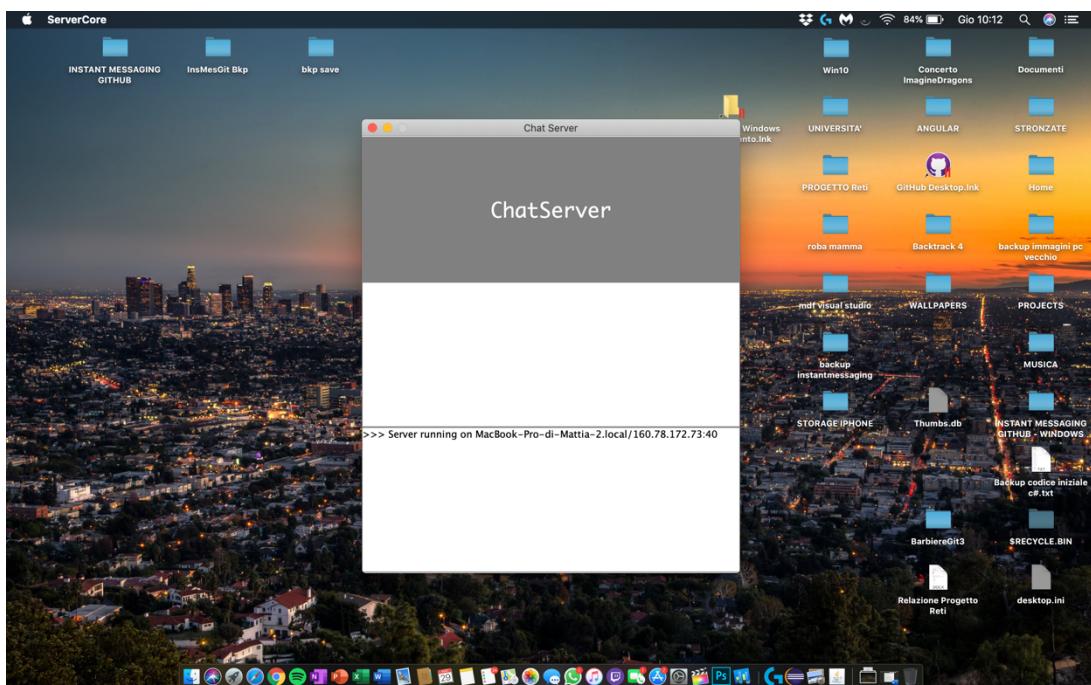
# IM Client – Server

Il programma consiste in una piattaforma di Instant-Messaging, basato su TCP, a cui un utente può connettersi per scambiarsi messaggi con un altro utente a scelta connesso alla stessa rete. La piattaforma si basa su un server che permette a tutti i client che si connettono di scambiarsi messaggi.

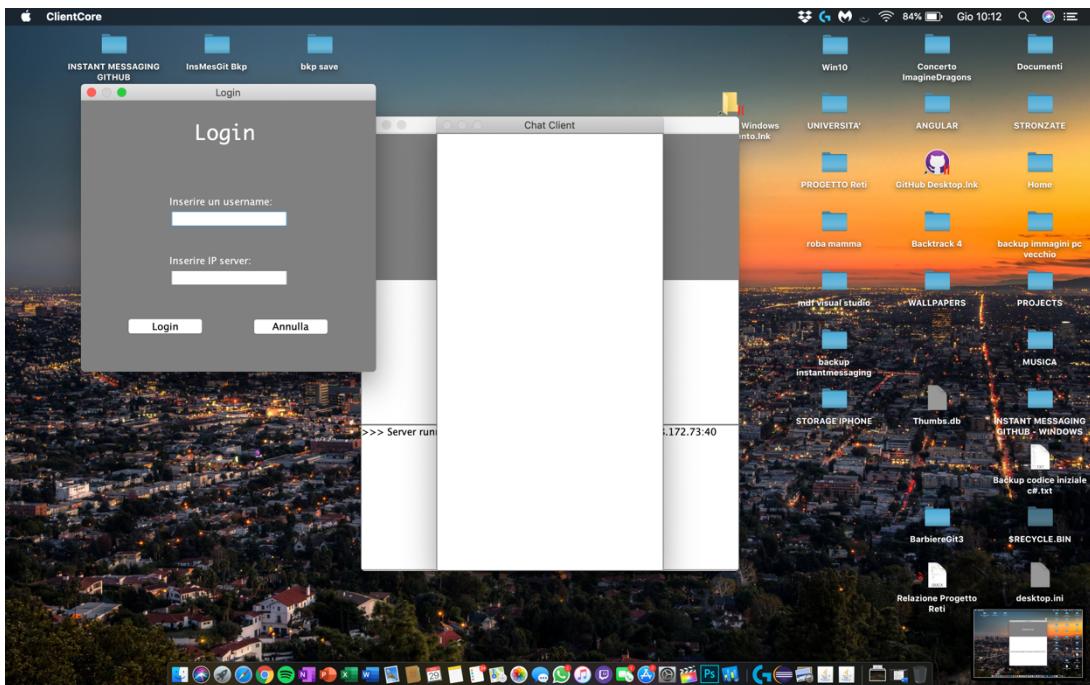
## Funzionamento

Avviamo il programma SERVER CORE.

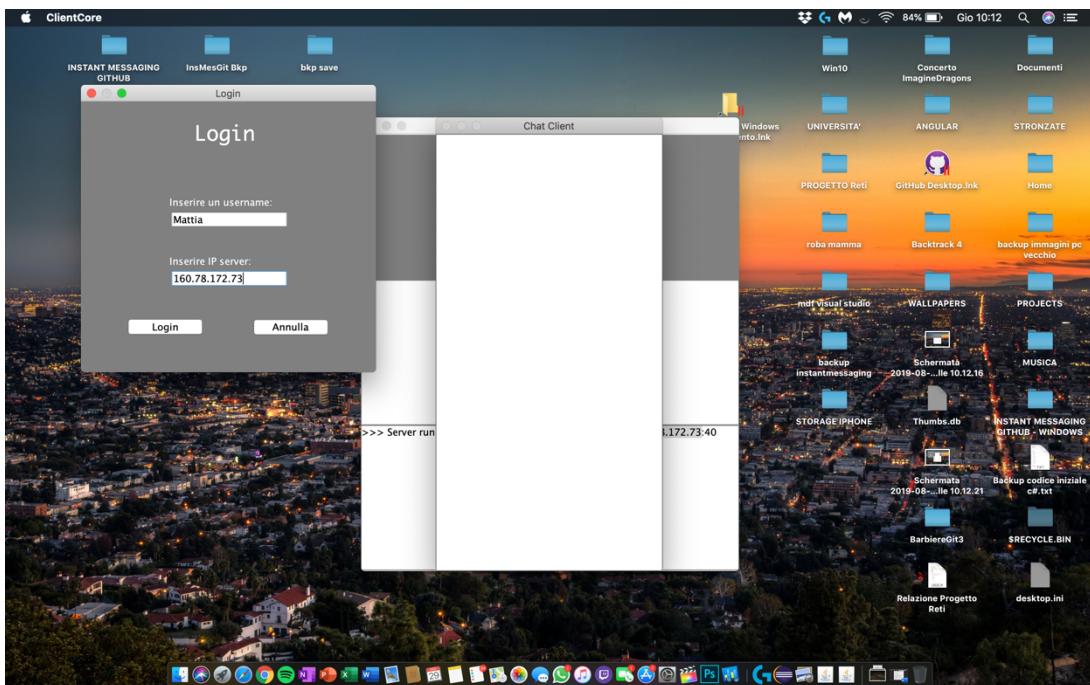
La console restituirà il nome della rete che stiamo utilizzando, il suo indirizzo che servirà agli utenti per fare accesso come client e il numero di porta utilizzato che sarà sempre 40.



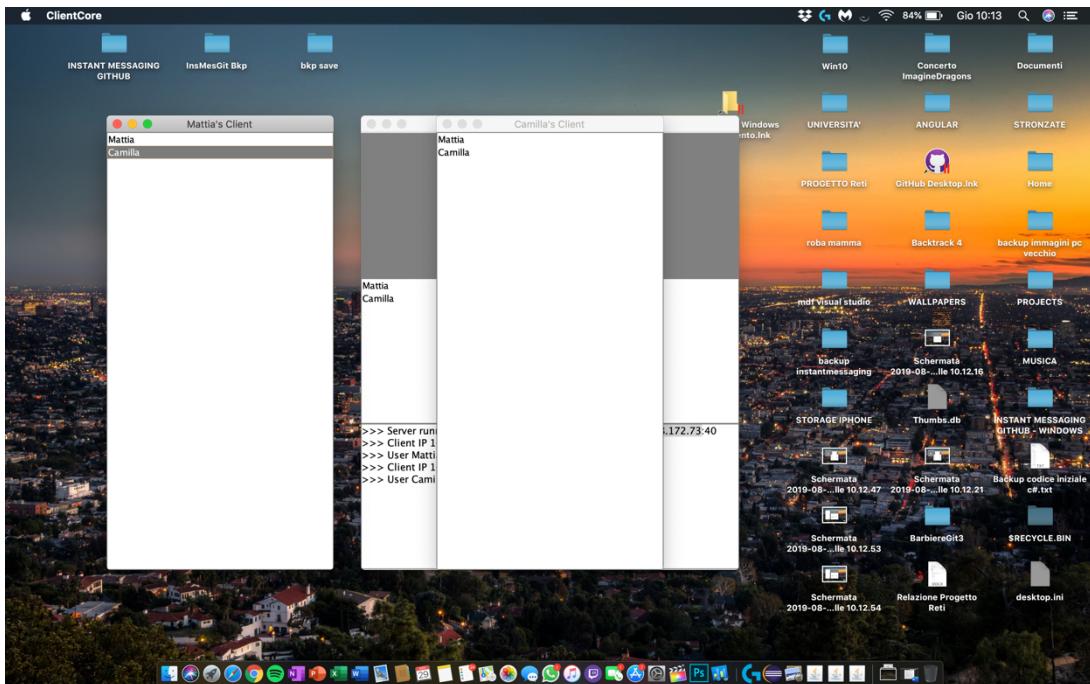
Successivamente avviamo il programma CLIENT CORE.



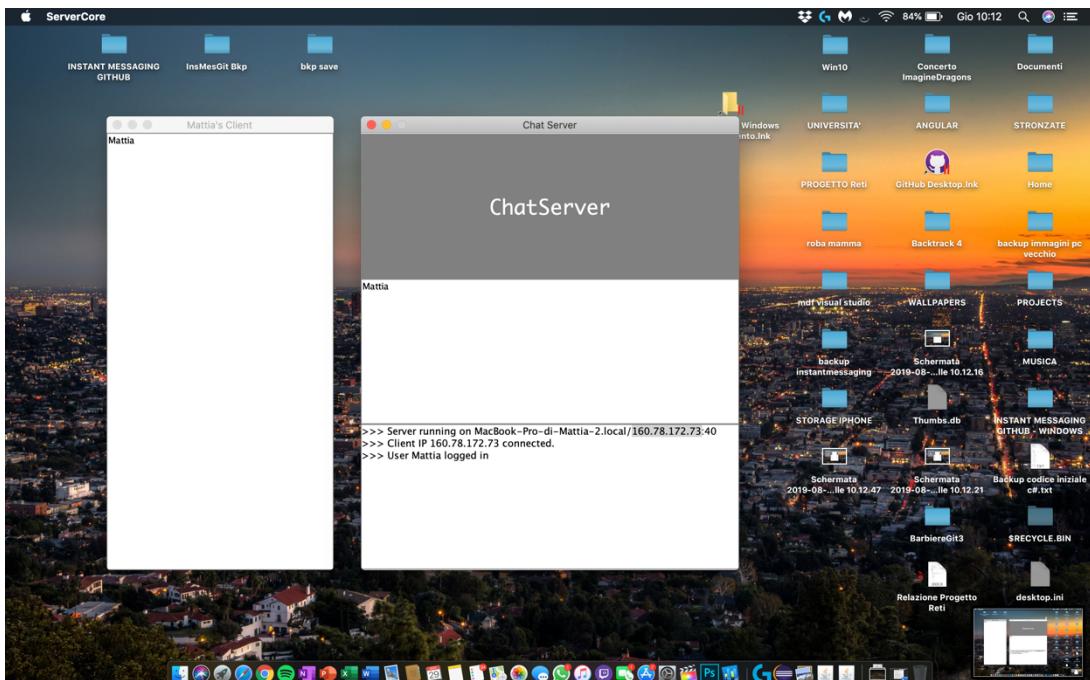
L'utente accederà al server scrivendo il proprio nome e l'indirizzo IP del server.



Fatto questo apparirà un menù con tutti gli utenti loggati con cui si potrà scegliere di iniziare una conversazione.

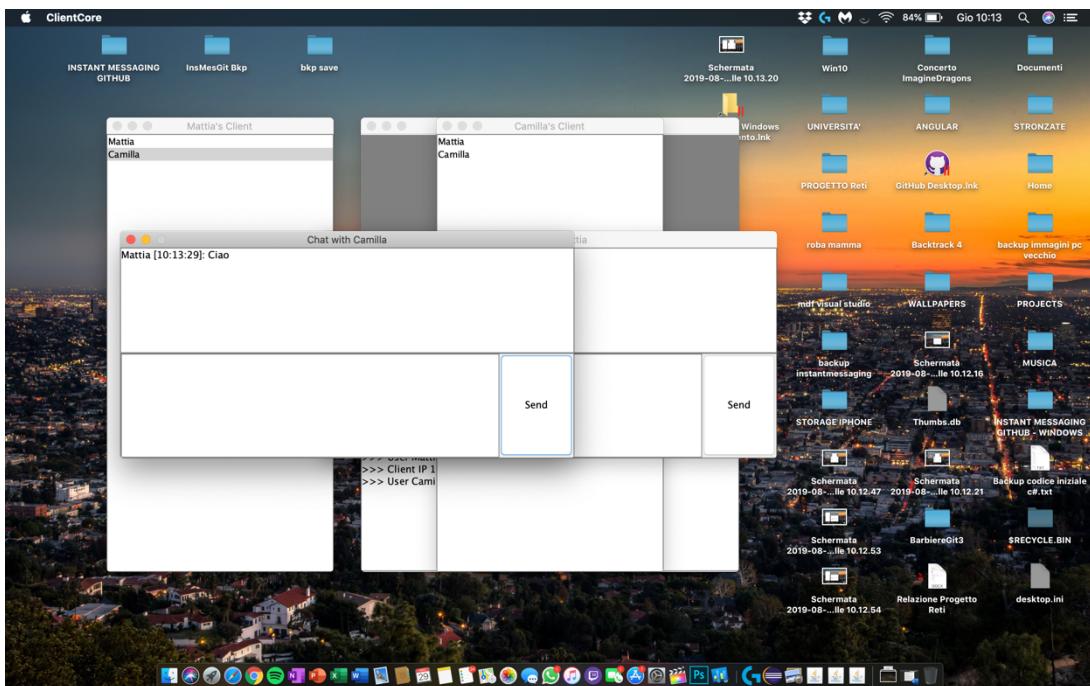


Ogni volta che un utente accederà al server, nella finestra del SERVER CORE apparirà un messaggio di login con il nome dell'utente che avrà effettuato la connessione.

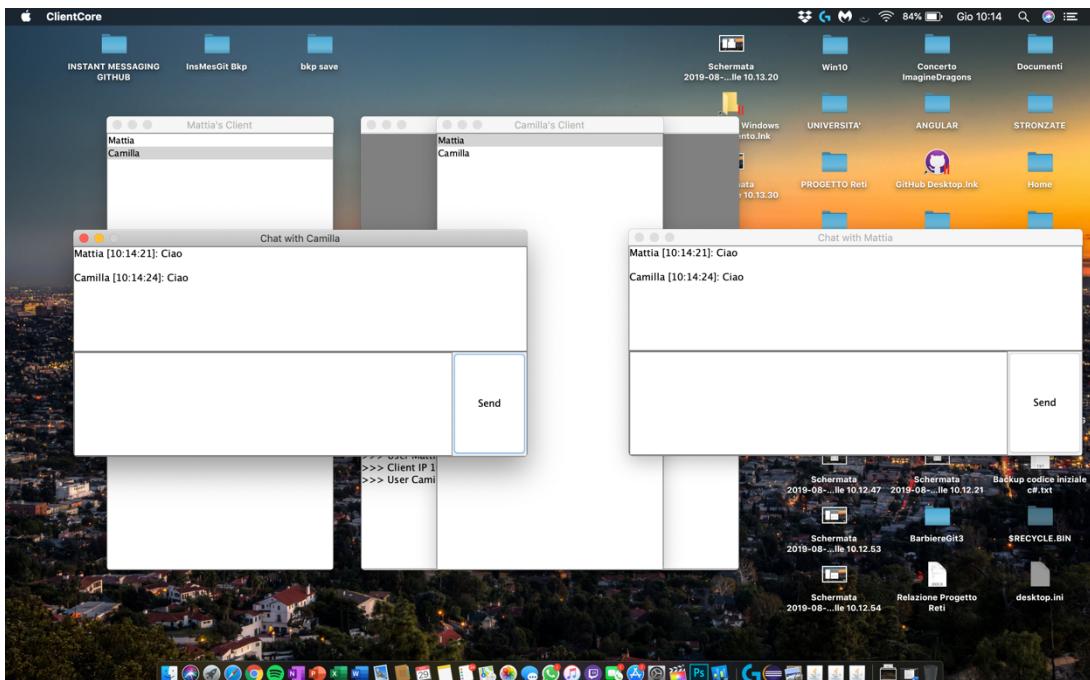


Dopo aver selezionato il nome dell'utente a cui si desidera inviare messaggi si aprirà una nuova finestra “Chat with (Nome Utente)”

all'interno della cui parte inferiore potremo scrivere i nostri messaggi da inviare.



L'invio avviene quando si clicca il pulsante “Send” posizionato sulla destra. I messaggi inviati vengono poi visualizzati nella parte superiore della chat insieme al nome del mittente e all'orario di invio del messaggio.



Per terminare la chat basta chiudere la finestra.

## Struttura del programma

Il programma si basa su 2 classi principali: ServerCore e ClientCore.

### ServerCore

La classe ServerCore è la prima che viene avviata per il corretto funzionamento del programma. Questa classe ha vari attributi:

- portNumber (int)  
Contiene il numero di porta su cui avviene la comunicazione ed abbiamo scelto come predefinita la porta 40;
- serverSocket (ServerSocket)  
Sono particolari tipi di socket che consentono la ricezione di richieste di servizio, quindi fornisce le funzionalità necessarie per la creazione del server che starà in ascolto su una determinata porta;
- IP (InetAddress)  
Oggetto che contiene sia l'host name che l'indirizzo IP;
- clientSocket (Socket)  
Classe per la gestione del client che effettua la chiamata;
- users (Map<String, CSHandler>)  
Utilizziamo la struttura dati Map per definire queste coppie di valori formate da un valore stringa che rappresentano i nomi degli utenti e l'oggetto CSHandler che si occupa dell'invio dei messaggi;

Apriamo una parentesi per parlare della classe CSHandler e della sua funzione. CSHandler rappresenta le azioni necessarie quando un messaggio viene inviato tra due client. Ad ogni client dev'essere assegnato un oggetto di tipo Handler.

Alla classe sono necessari solo 3 attributi:

- socket (Socket);
- objIn (ObjectInputStream);
- objOut (ObjectOutputStream).

In particolare queste ultime due classi, ObjectInputStream e ObjectOutputStream, sono classi che definiscono flussi di dati su cui si possono leggere e scrivere oggetti.

Nel costruttore gli attributi vengono inizializzati con i valori inseriti e poi viene eseguito il metodo start() che richiama il metodo run().

All'interno del metodo run() vengono gestiti i messaggi in entrata in base al tipo. Viene creato un oggetto Message e uguagliato al messaggio che verrà letto tramite il metodo readObject() presente nella classe ObjectInputStream. Se il messaggio ricevuto è di tipo MESSAGE verrà inserito nella lista di messaggi da recapitare e se invece è di tipo

LOGOUT allora viene richiamato il metodo removeUser() che prende in input il nome dell'utente da rimuovere estrapolandolo tramite la funzione getSender() dal messaggio appena ricevuto. Alla fine vengono chiusi gli stream con la funzione close().

L'altro metodo importante è il sendMessage() che riceve in input un oggetto di tipo Message e poi tramite la funzione writeObject() della classe ObjectOutputStream viene inviato all'utente per cui si richiamerà questa funzione.

- msgQueue (Queue<Message>)

Contiene i messaggi inviati e abbiamo scelto di utilizzare una queue per la politica di accesso FIFO (first in, first out);

Approfondiamo la classe Message. Prima degli attributi della classe vengono definiti dei valori interi static final che ci serviranno per definire il tipo del messaggio. La classe contiene due attributi di tipo String per il nome del mittente e il nome del destinatario, poi contiene un attributo di tipo Object chiamato data che rappresenta il contenuto del messaggio e infine un attributo di tipo int. Il valore che può assumere Type corrisponde a uno dei valori static final che abbiamo definito in precedenza e così siamo subito in grado di capire a cosa servirà il messaggio recapitato.

```
//Define type of message
public static final int MESSAGE = 0;
public static final int LOGIN = 1;
public static final int LOGOUT = 2;
public static final int UPDATE_USERS = 3;
public static final int SERVER_RESPONSE = 4;
```

- serverListener (Listener)

Questa classe si occupa della gestione dei messaggi all'interno di msgQueue.

La classe Listener contiene due metodi:

- run()

Attraverso un while(true) controlla lo stato di msgQueue. Verifica per prima cosa che non sia vuota e nel caso lo fosse richiama il metodo wait(), altrimenti se contiene dei messaggi crea un nuovo oggetto di tipo Message chiamato currentMsg che diventa uguale al primo elemento della coda tramite il metodo poll(). Tramite un for() che scorre la chiave dell'attributo users poi cerca il destinatario del messaggio e richiama poi la funzione sendMessage della classe CSHandler;

- addMsg(Message message)

Riceve in input un oggetto di tipo Message che viene poi aggiunto a msgQueue tramite il metodo add().

Gli ultimi due attributi della classe ServerCore contengono le informazioni riguardanti la grafica della console e la sua struttura:

- design (ChatSDesign)

E' un oggetto della classe ChatSDesign che contiene la funzione updateUsers() che ricevendo in input un array di stringhe si occupa della visualizzazione a video di tutti i nomi degli utenti con cui è possibile instaurare una conversazione;

- controller (CSController)

E' un oggetto della classe CSController che contiene la funzione writeMsg() che prende in input una stringa e la fa apparire all'interno della finestra server.

Continuiamo l'analisi della classe ServerCore con il costruttore. Dopo le varie definizioni degli attributi viene richiamata la funzione init() che si occupa dell'inizializzazione del server. In questa funzione, tramite il metodo getLocalHost() viene copiato nella variabile IP della classe l'indirizzo di rete a cui gli utenti devono collegarsi per scambiarsi i messaggi. E' necessario che gli utenti siano tutti collegati alla stessa rete per chattare. Proseguendo viene creato il serverSocket dando in input il numero di porta su cui deve avvenire la comunicazione. Richiamiamo poi controller per scrivere a console le informazioni relative al server: il nome, l'indirizzo IP e la porta. A questo punto il metodo init() è finito e nel costruttore viene istanziato un oggetto Listener chiamato serverListener che si occupa dello stream dei messaggi. Per ultimo, all'interno di un while(true), viene richiamato il metodo listen() della classe principale ServerCore: il server si mette in ascolto e viene istanziato un oggetto clientSocket con la socket che il server trova tramite la funzione accept() della classe ServerSocket. Viene stampata a video la conferma dell'avvenuta connessione dell'utente e poi chiama la funzione loginUser() che prende in input la socket appena creata. In loginUser() utilizziamo le classi ObjectInputStream e ObjectOutputStream per scrivere i messaggi di login a console. Inizializziamo objIn e objOut come nuove istanze delle due classi per la lettura e la scrittura di oggetti, poi creiamo un nuovo oggetto di tipo Message chiamato loginRequest che sarebbe il primo messaggio letto tramite la funzione readObject(). Dopo aver controllato che il message loginRequest sia di tipo LOGIN e che l'attributo Data sia diverso da vuoto, facciamo un controllo sul nome utente inserito dal client. Facciamo un controllo scorrendo l'attributo users e controllando le key

presenti: se troviamo una key di users uguale all’attributo Data di loginRequest (che contiene l’user name) viene incrementata una variabile i. Alla fine dello scorrimento della struttura dati users, la variabile i ci offre due possibilità: se questa è maggiore di 0 significa che è già presente un utente con l’username appena inserito quindi non è possibile proseguire e viene mandato un messaggio di tipo SERVER.RESPONSE con l’attributo Data uguale a “USERNAME\_ERROR”, altrimenti se i è uguale a 0 creiamo un nuovo Handler, aggiungiamo l’utente all’hashmap della chat e scriviamo a console la conferma dell’avvenuto login. Inoltre creiamo un messaggio di tipo SERVER\_RESPONSE con la frase “Success” e lo invia al server. Richiamiamo il metodo updateAllUsers() che comunica a tutti gli utenti connessi l’entrata nella chat di un nuovo membro: questa funzione tramite un for che scorre tutte le chiavi dell’attributo users manda a ognuno un messaggio dal server di tipo UPDATE\_USERS. Il messaggio inviato nell’attributo data contiene la funzione getUsernames() che crea un array per scrivere l’elenco degli utenti. Infine l’elenco degli utenti nella grafica del server viene aggiornato con la funzione updateUsers(). Se una delle due condizioni iniziali non è rispettata viene inviato un messaggio al server di tipo SERVER\_RESPONSE per indicare la non riuscita dell’operazione di login.

L’ultima funzione che analizziamo della classe ServerCore è removeUser() che prende in input una stringa con il nome dell’utente da eliminare: utilizziamo il metodo remove() per togliere user dalla Map users, richiamo la funzione writeMsg() dell’attributo controller per stampare sulla console il messaggio di disconnessione, uso la funzione updateAllUsers() per aggiornare l’elenco utenti e per ultimo con la funzione updateUsers() dell’attributo design viene stampato l’elenco aggiornato a video.

## ClientCore

L’altra classe del programma è clientCore e si occupa della gestione della parte client. Analizziamo gli attributi:

- portNumber (int)  
Indica il numero della porta su cui avviene la comunicazione;
- hostName (string)  
Indica l’indirizzo IP del client;
- view (ChatCDesign)  
Si occupa della creazione della struttura della finestra;
- cObjIn (ObjectInputStream); cObjOut (ObjectOutputStream)

- Si occupano di lettura e scrittura degli oggetti sugli stream;
- socket (Socket)  
Si occupa del canale di comunicazione;
- username (String)  
Nome del client
- chats (HashMap<String, Chatwindow>)  
Coppie chiave-valore che contengono il nome utente e la chat-console corrispondente;
- msgQueue (Queue<Message>)  
Coda dei messaggi che segue il principio FIFO.

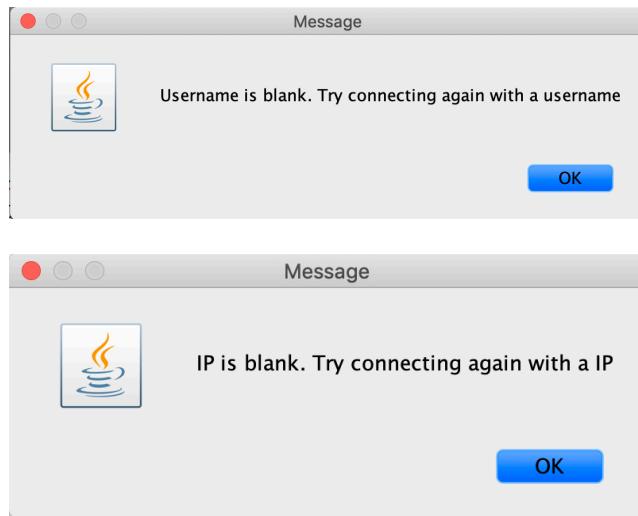
Iniziamo ad analizzare il costruttore. Viene creato un oggetto di tipo ChatCDesign chiamato view per la creazione della finestra del client. Richiama la funzione connect(): all'interno di questa crediamo una socket che ha come input l'indirizzo IP di host-name e il numero di porta di portNumber. Creiamo due nuovi oggetti di tipo ObjectInputStream e ObjectOutputStream che associamo al socket appena creato: questi ci servono a definire l'input e l'output sull'indirizzo.

Sempre all'interno di connect() effettuiamo il login scrivendo tramite il metodo writeObject() dell'oggetto cObjOut un nuovo message di tipo LOGIN che ha valore null come mittente e come destinatario.

Creiamo poi un nuovo oggetto Message chiamato confirmation che eguagliamo al messaggio che viene letto tramite cObjIn.readObject(). Questo messaggio in entrata contiene l'informazione riguardante il login effettuato prima, cioè se è andato a buon fine oppure no. Con un if controlliamo il contenuto dell'attributo data di confirmation: se il login ha avuto successo lo stampiamo a console, poi inizializziamo un nuovo oggetto Message chiamato users che verrà eguagliato al messaggio in entrata letto tramite readObject() e che sarà di tipo UPDATE\_USERS e servirà per aggiornare l'elenco utenti disponibili. Se l'username inserito dall'utente è già in uso allora il Message confirmation avrà attributo Data uguale a "USERNAME\_ERROR", quindi utilizziamo la classe JOptionPane per creare un pop-up e stampare un messaggio di errore con il metodo showMessageDialog.



Stessa cosa avverrà se, all'interno del login, non saranno stati inseriti username o IP.



Tornando al costruttore, dopo la chiamata della funzione connect() viene inizializzata una linked list di Message chiamata msgQueue e rappresenta la lista dei messaggi che il client riceve. Poi creiamo una struttura dati hashmap formata dalla coppia <string, ChatWindow> chiamata chats che contiene l'elenco dei nomi degli utenti e le corrispondenti finestre di chat. Per ultimo creiamo un oggetto di tipo Listener che chiamiamo chatListener e invochiamo il suo metodo start(). Il metodo run() richiamato da start() si occupa di aggiungere i messaggi a msgQueue e poi li analizza tramite il principio FIFO dando in input msgQueue.poll() alla funzione parseMessage().

ParseMessage() tramite uno switch() gestisce 2 tipi di messaggi dividendoli in base all'attributo type:

- message.MESSAGE  
Viene chiamata la funzione receiveMessage();

Nella funzione receiveMessage() effettuiamo il controllo se il sender è presente nell'hashmap chats:

- Se è presente stampa il sender, la data e il messaggio;
- Se non è presente prima apre una nuova chat e poi stampa il sender, la data e il messaggio.

- Message.UPDATE\_USERS

Aggiorna l'elenco degli utenti con updateUsers().