Tugas Akhir Semester
Computer Network

Dibuat Oleh :

NIM 11106055 Mario Simaremare

NIM 11106040 Edison Sihotang

Untuk :
Politeknik Informatika Del
Laguboti

Daftar Isi

Daftar Gambar

1 Fitur

Dalam bab ini akan jelaskan mengenai fitur-fitur yang ada dalam aplikasi chating.

1.1 Server

Berikut tugas-tugas *Server* dan hal-hal yang dapat dilakukannya:

1. Menerima permintaan koneksi dari *client* dan memberikan identitas yang unik untuk masing-masing koneksi baru.
2. Mendaftarkan *client* sebagai *client* yang *on-line.*
3. Menjadi jembatan antar-*client* dalam berkomunikasi, baik pesan yang ditujukan untuk public maupun yang rahasia (private), pesan-pesan koneksi (*client offline*, *client* baru memasuki ruang *public*).
4. Mengusir *client* (dengan memutuskan koneksi secara sepihak) dari ruang public.

1.2 Client

Berikut hal-hal yang dapat diakukan *Client* :

1. *Client* dapat bergabung ke ruang pembicaraan *public* dengan terlebih dahulu mendaftarkan diri ke *Server* tertentu, yang menyediakan layanan *chatting*.
2. *Client* dapat mengirim pesan ke semua *client* lain yang sedang *online*.
3. *Client* dapat mengirim pesan private ke *client* lainnya yang sedang *online*.

2 Message Format

Pada bab ini dijelaskan mengenai format pesan yang digunakan. Tujuan pemformatan pesan adalah memudahkan proses merutekan pesan ke *client* yang sesuai.

Sebagai contoh, pesan yang ditujukan untuk umum (*public messages*) harus dikirimkan ke semua *online client*, namun tidak berlaku dengan pesan yang rahasia (*private message*) hanya akan dikirmkan ke penggua yang tepat.

2.1 Format Pesan dari Server

Berikut format pesan yang diimplementasikan oleh *server* :

1. `REMOVE user_id user_name.`

   Format pesan ini digunakan *server* untuk memberitahu semua *online client* bahwa *client* yang *id* dan *username*-nya tertera pada pesan telah meninggalkan ruang pebicaraan umum (*public*

room chat) dengan alasan yang tidak jelas (koneksi terputus tanpa sebab yang jelas).

2. `KICKED`

   Format pesan ini digunakan *server* untuk memberitahu *client* bahwa ia telah dikeluarkan dari ruang pembicaraan umum oleh *server.*

3. `DISCONNECT`

   Format pesan yang digunakan oleh *server* untuk memberitahu semua *online client* bahwa *server* akan segera *offline*.

2.2 Format Pesan dari Client

Berikut format pesan yang diimplementasikan oleh *client* :

1. `REGISTER user_name`

   Format pesan ini digunakan *client* untuk mendaftarkan diri ke ruang pembicaraan umum.

2. `REMOVE user_id user_name`

   Format pesan ini digunakan *client* untuk memberitahu semua *online client* bahwa *client* yang *id* dan *username*-nya tertera pada pesan (si pengirim pesan) akan segera meninggalkan ruang pebicaraan umum (*public room chat*).

3. `TELL destination_user_id user_id user_name`

   Format pesan ini digunakan *client* untuk memberitahu *online client* lainnya bahwa ia telah terkoneksi dan bergabung di ruang pembicaraan umum (*public room chat*).

4. `CHAT PUBLIC user_id user_name>>pesan`

   Format pesan ini digunakan *client* untuk mengirim pesan keseluruh *online client* (*public messages*).

5. `CHAT PRIVATE destination_id user_id user_name>>pesan`

   Format pesan ini digunakan *client* untuk mengirim pesan rahasia pada seorang *online client* (*private messages*).

3 Skenario

Dalam bab ini akan dijelaskan mengenai cara kerja dari *server* dan *client*. Bagaimana *server* mengawali layanan, menerima lebih dari satu koneksi dan sebagai penghubung antar-*client*.

3.1 Skenario di sisi Server

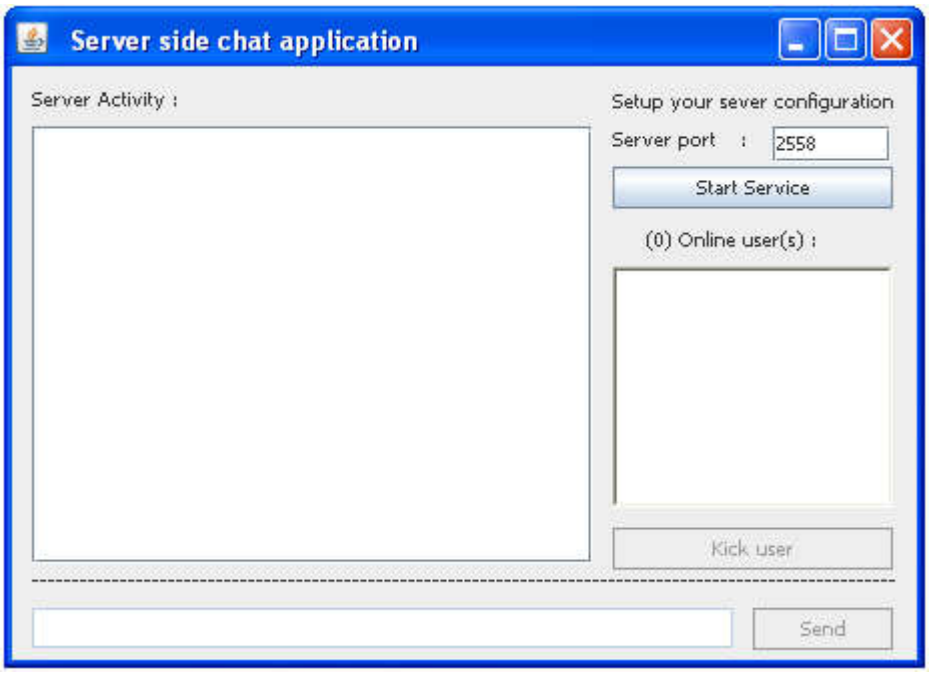
Dalam kasus ini terdapat dua kelas yang digunakan oleh *server*. Pertama adalah kelas Server dan yang kedua adalah kelas ServerThread.

3.1.1 ServerGUI

Kelas ServerGUI adalah kelas yang menangani antarmuka antara *Administrator server* dengan aplikasi *server* yang menyediakan layanan *multichat*.

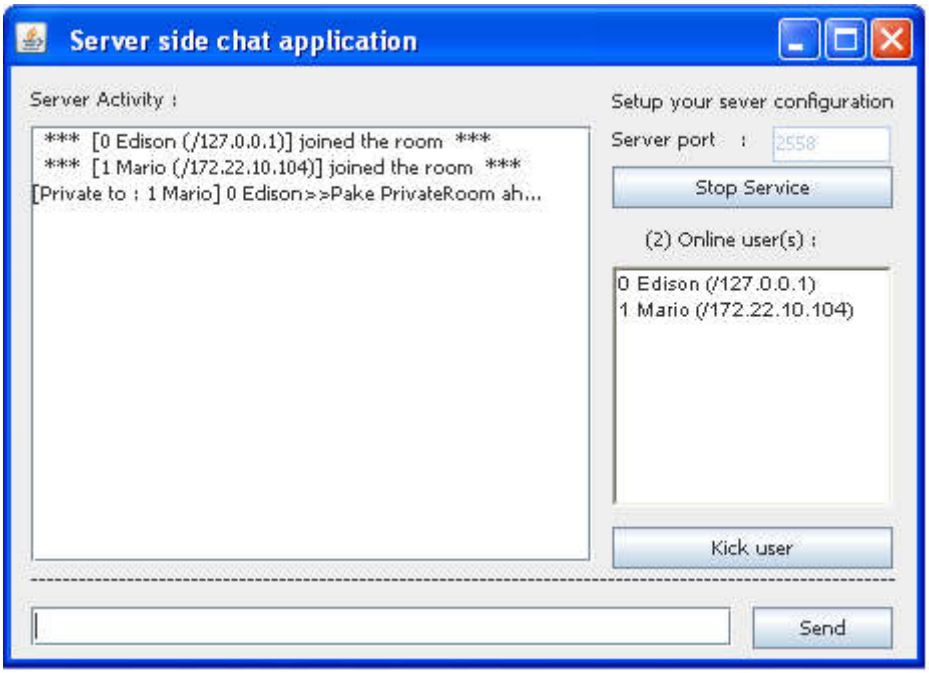
Skenario kerja dari kelas Server :

1. Tampil window ServerGUI. Administrator harus terlebih dahulu mengkonfigurasi *port* dari server, di mana server akan menerima koneksi.



Gambar 1. Starting ServerGUI

2. *Administrator* dapat melakukan pengusiran dan memonitor aktivitas pengguna layanan.



Gambar 2. Monitoring or kicking ServerGUI

3. *Administrator* juga dapat mengirimkan pesan keseluruh *online client*.

3.1.2 Server

Kelas Server adalah kelas dasar yang akan menyediakan layanan *multichat*. Kelas ini akan menerima *request* koneksi dari *client*, mendaftarkan *client* ke ruang pembicaraan umum serta menjamin keterhubungan antar-*client*.

Skenario kerja dari kelas Server :

1. *Server* membangkitkan layanan *multichat* dan siap menerima *request* koneksi dari *client*.
2. *Server* menerima *request* koneksi dan menugaskan sebuah objek socket yang akan menangani koneksi yang baru terbentuk.
3. *Server* akan mengirimkan identitas unik (*id*) yang akan membedakan setiap koneksi yang terbentuk.
4. *Server* membentuk sebuah objek ClientConnection yang akan berdiri sendiri (*independent*) dan menangani koneksi secara terpisah dari *server*, namun tetap mengacu pada *server*. Dalam pembentukan objek ini akan di-*parsing* objek Socket yang menyimpan koneksi dan objek Server.

5. *Server* akan mendaftakan objek ClientConnection jika *client* telah memintanya.
6. *Server* kembali siap menerima koneksi baru.
7. *Server* dapat mengirim pesan dari sebuah ClientConection keseluruh *online client* ataupun ke seorang *client*, dalam hal ini *client* yang terdaftar, sesuai dengan kebutuhan masing-masing objek ClientConnection. Dalam kasus ini Server bertindak sebagai penghubung antar-ClientConnection, karena masing-masing ClientConnection tidak dapat saling berhubungan secara langsung dan berjalan secara *independent*.

3.1.3 ClientConnection

ClientConnection adalah kelas yang akan menangani sebuah koneksi *Server-Client* dan bertugas meneruskan pesan pada Server untuk di-*broadcast* ke *client* lainnya (*messaging*).

Skenario kerja dari ClientConnection :

1. Ketika objek *ClientConnection* terbentuk, *ClientConnection* menerima objek referensi socket (yang menyimpan koneksi baru), referensi objek Server (induk dari semua proses yang terjadi pada sisi *server*).
2. Objek *ClientConnection* akan berjalan secara *independent* dan akan memelihara koneksi dengan *client*, bertindak sebagai penghubung antara *server* dan *client*.
3. Objek *ClientConnection* siap menerima pesan dan perintah dari *client* yang akan dikomunikasikan dalam koneksi antar-*client.*
4. Ketika sebuah pesan maupun perintah yang dikirim dari *ServerConnection* (objek yang menangani koneksi di sisi *client*) diterima, objek *ClientConnection* akan memeriksa *keyword* yang disertakan. Setiap *keyword* memiliki arti perintah yang berbeda-beda. Berdasarkan *keyword* tersebut *ClientConnection* memutuskan aksi apa yang harus dilakukan.
5. Seterusnya *ClientConnection* akan menunggu pesan maupun perintah yang datang dari sisi *client*.

3.2 Skenario di sisi Client

Disisi client diperlukan tiga kelas, masing-masing:

1. ServerConnection,

   Sebagai kelas yang menangani komunikasi *client* via jaringan secara *background*.

2. PublicRoomChat,

   Kelas yang menangani antarmuka antara pengguna dan ServerConnection. Pada kelas ini pengguna melakukan *setting*

awal koneksi, melakukan pengiriman pesan yang akan dikirimkan ke semua *online client*.

3. PrivateRoomChat,

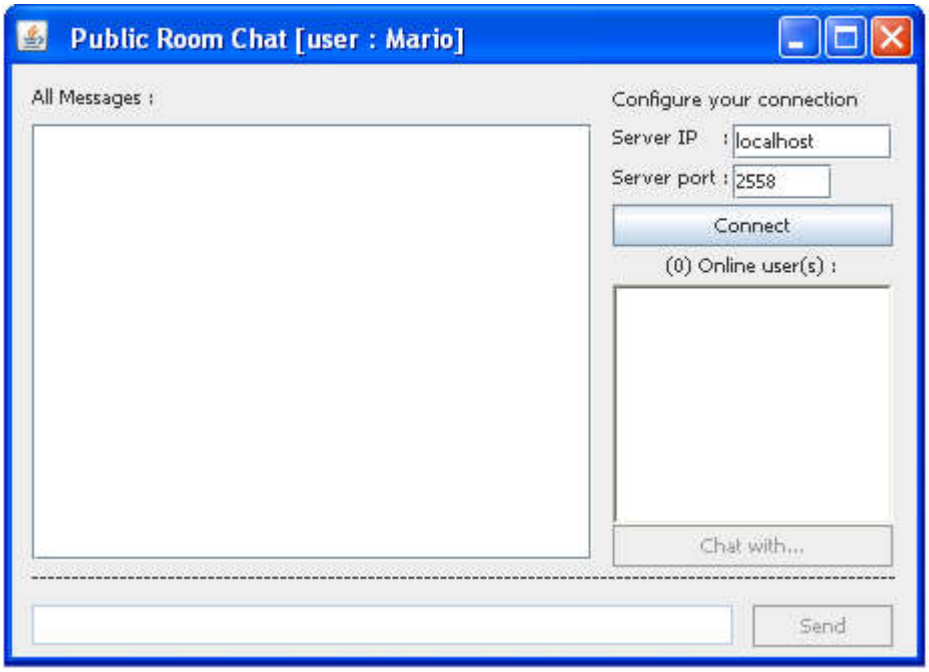
   Kelas yang khusus menangani antarmuka percakapan private.

3.2.1 PublicRoomChat :

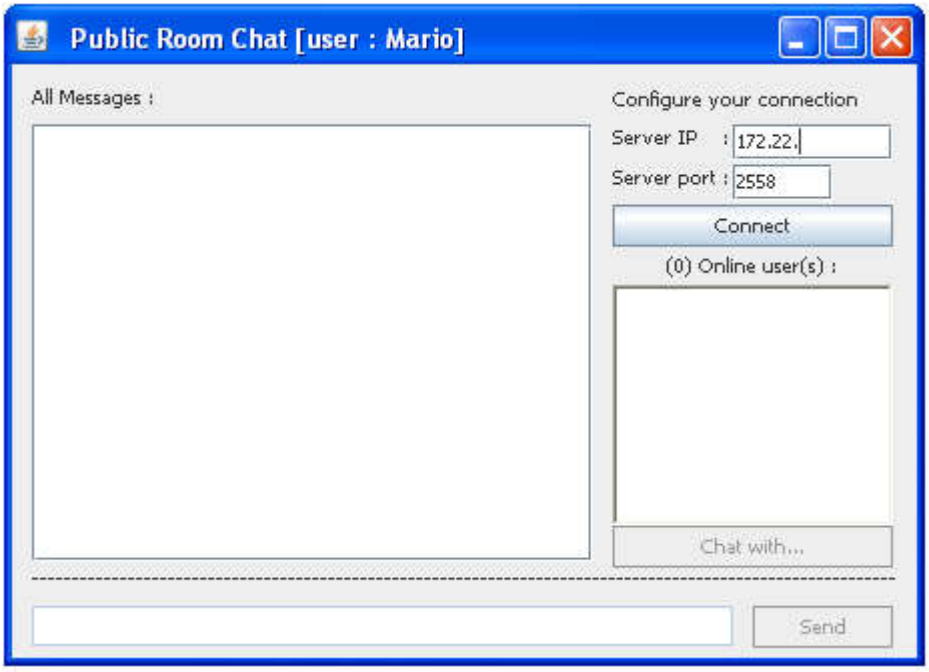
1. Tampil *input dialog*, di sini pengguna harus memasukan nama sebagai *username*.

Gambar 3.Input dialog ClientGUI

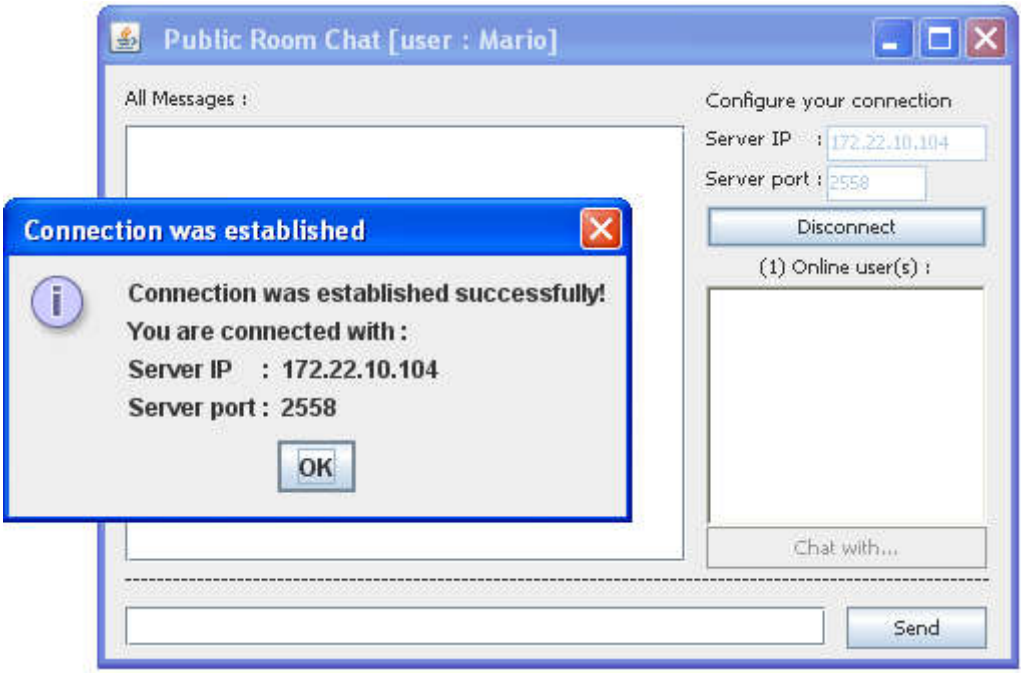
2. Setelah pengguna memasukan nama, akan tampil *window public Room Chat*. Selanjutnya pengguna harus mengkonfigurasi koneksi (telah diberikan nilai *default*) dan melakukan koneksi dengan menekan tombol "*Connect*". Pada saat itu juga terinisialisasi objek ServerConnection.



Gambar 4. Tampilan awal PublicRoomChat



Gambar 5. Konfigurasi PublicRoomChat

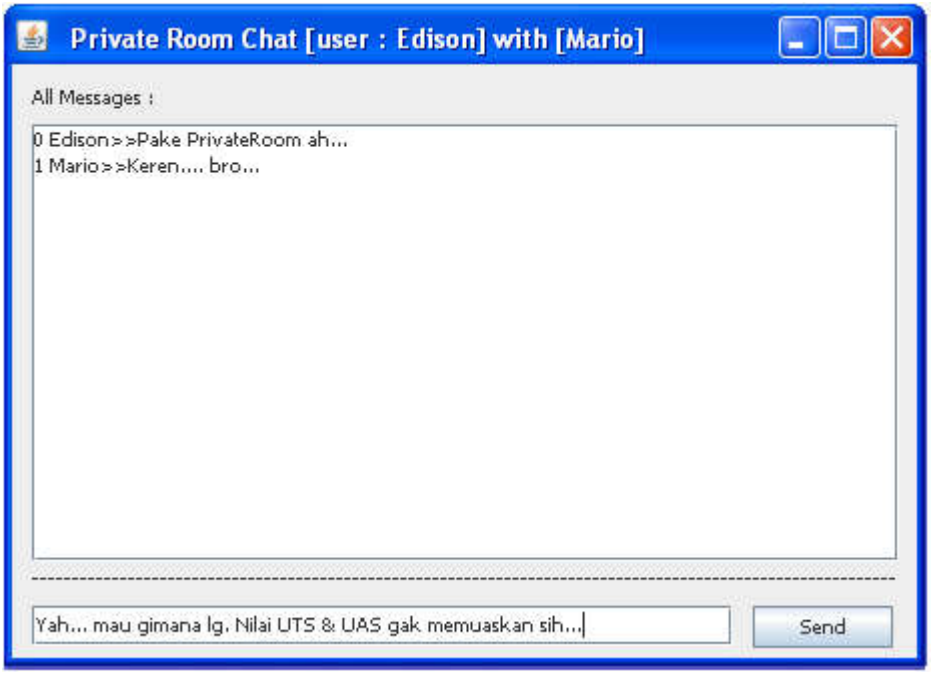


Gambar 6. Terkoneksi dengan server

3. Setelah koneksi tebentuk dan berhasil, pengguna akan terdaftar sebagai *online client* dan dapat melakukan *chat* dengan *online client* lainnya. Pengguna juga dapat melakukan pembicaraan privat dengan *client* yang terdaftar dan sedang *online* pada *online users*.
4. Objek PublicRoomChat juga akan me-*route* pesan yang bersifat privat ke objek PrivateRoomChat yang tepat.

3.2.2 PrivateRoomChat

1. Objek PrivateRoomChat akan terbentuk jika: *client* menginisialisasi percakapan dengan *online client* lainnya, dengan memilih *online client* yang ada pada *online user* atau *client* sedang menerima pesan yang bersifat privat dari *online client* lainnya.



Gambar 7. Pesan one to one

2. Objek ini akan menangani percakapan yang bersifat privat dengan *online client* tertentu.

3.2.3 ServerConnection

1. Objek ini bertanggungjawab atas koneksi dengan ClientConnection yang berada di sisi *server*. Objek ini akan terbentuk ketika pengguna menekan tombol "*Connect*", dan akan di-*destroy* ketika koneksi putus, *client* diusir dari ruang percakapan umum, atau client menekan tombol "*Disconnect*".
2. Objek ini akan menunggu pesan maupun perintah yang datang dari ClientConnection di sisi server kemudian menerjemahkannya menjadi aksi-aksi yang sesuai dengan *keyword* yang disertakan.
3. Objek ini juga sebagai pengkode (memberi format pada pesan) sebelum mengirimkannya.

4 Implementasi

Pada bab ini akan dijelaskan implementasi yang dilakukan untuk mendukung aplikasi, baik di sisi *server* maupun *client*.

4.1 Implementasi di sisi server

Di bawah ini merupakan implementasi di sisi *server*.

4.1.1 Implementasi Server

4.1.1.1 Membangun Server

Berikut bagian kode yang akan menerima koneksi.

```
public void run(){
  try{
    serverSocket = new ServerSocket(serverPort);
    System.out.println(" This server is listening on : " +
      serverPort);
    serverGUI.serviceStartedSuccessfully();
    running = true;

    while(running){
      socket = serverSocket.accept();
      ia = socket.getInetAddress();
      addConnection();
    }
  }catch(IOException ioe){
    if(running){
      serverGUI.startServiceFailed();
      System.out.println("\n An IOException occured! (" + ioe + ")");
    }
  }catch(Exception e){
    if(running){
      serverGUI.startServiceFailed();
      System.out.println("\n An Exception occured! (" + e + ")");
    }
  }
}
```

4.1.1.2 Penyimpan Koneksi (Collection)

Berikut potongan kode yang akan menambahkan objek ClientConnection, yang bertugas menangani koneksi *client* di sisi server.

```
synchronized void setConnectionTable(String _id,
  ClientConnection _CC){

  connectionTable.remove(_id);
  connectionTable.put(_id, _CC);
}
```

Berikut potongan kode yang akan menghapus objek ClientConnection, penghapusan dilakukan jika *client offline*.

```
synchronized void setConnectionTable(
  String _id, ClientConnection _CC){
```

```
  connectionTable.remove(_id);
  connectionTable.put(_id, _CC);
}
```

4.1.1.3 Pengiriman Pesan Publik

```
synchronized void broadcast(String _id, String _message){
  watchUserActivity(_id, _message);
  Enumeration e = connectionTable.keys();
  String other = new String();

  while(e.hasMoreElements()){
    other = (String) e.nextElement(); // read the keys one by one
    ClientConnection CC =
     (ClientConnection) connectionTable.get( other);

    if(!other.equals(_id)){
      CC.write(_message);
    }
  }
}
```

4.1.1.4 Pengiriman Pesan Privat

```
synchronized void send(String _id, String _message){
  watchUserActivity(_id, _message);
  System.out.println(_id + ":" + _message);
  ClientConnection CC = (ClientConnection) connectionTable.get( _id);
  CC.write(_message);
  System.out.println("_id:" + _id + " | _message" + _message);
}
```

4.1.2 Implementasi ClientConnection

4.1.2.1 Implementasi Pembacaan Pesan dari Client

```
private String read(){
  try{
    return(in.readLine());
  }catch(SocketException se){
    System.out.println("\n An SocketException occured! (" +
      se + ")");
    server.kill(this);
    return (null);
  }catch(Exception e){
    System.out.println("\n An Exception occured! (" + e + ")");
    server.kill(this);
    return (null);
  }
}

public void write(String _message){
  try{
    out.println("" + _message);
  }catch(Exception e){
    System.out.println("\n An Exception occured! (" + e + ")");
  }
}

// bellow are methods in maintain connection.
```

```
public void close(){
  server.kill(this);
  try{
    socket.close();
  }catch(IOException ioe){
    System.out.println("\n An IOException occured! (" + ioe + ")");
  }
}

// bellow are needed attributes and methods while this object is
running.

private static final int ADD = 0;
private static final int REMOVE = 1;
private static final int CHAT = 2;
private static final int PUBLIC = 3;
private static final int TELL = 4;
private static final int REGISTER = 5;
private static Hashtable keys = new Hashtable();
private  static  String  keystring[]  =  {"ADD",  "REMOVE",  "CHAT",
"PUBLIC", "TELL", "REGISTER"};
static {
  for(int i = 0; i < keystring.length; ++i){
    keys.put(keystring[i], new Integer(i));
  }
}

private int lookup(String _s){
  Integer i = (Integer) keys.get(_s);
  return (i == null ? -1 : i.intValue());
}

// bellow are overiden methods

public void run(){
  server.setConnectionTable(id, this);

  String s;
  StringTokenizer st;
  String keyword;

  while((s = read()) != null){
    System.out.println(id + "CC:" + s);
    st = new StringTokenizer(s);
    keyword = st.nextToken();
    switch(lookup(keyword)){
      case ADD:{
        server.broadcast(id, s);
        break;
      }
      case REMOVE:{
        server.kill(this);
        break;
      }
      case CHAT:{
        keyword = st.nextToken();
        if(lookup(keyword) == PUBLIC){
          String _message = st.nextToken();
          while(st.hasMoreTokens()){
            message = _message + " " + st.nextToken();
          }
```

```
          server.broadcast(id, "CHAT PUBLIC " + _message);
        }else{
          String _otherID = st.nextToken();
          String _message = st.nextToken();
          while(st.hasMoreTokens()){
            _message = _message + " " + st.nextToken();
          }
          server.send(_otherID, "CHAT PRIVATE " + id + " " + name +
            " " + _message);
        }
        break;
      }
      case TELL:{
        String _id = st.nextToken();
        String _message = st.nextToken();
        while(st.hasMoreTokens()){
          _message = _message + " " + st.nextToken();
        }
        server.send(_id, _message);
        break;
      }
      case REGISTER:{
        String _name = st.nextToken();
        while(st.hasMoreTokens()){
        _name = _name + " " + st.nextToken();
        }
        setName(_name);
        break;
      }
      default:{
        System.out.println(s);
      }
    }
  }
}
```

4.2 Implementasi di sisi Client

4.2.1 Implementasi PrivateRoomChat

Kelas ini merupakan kelas yang menghubungkan mesin dengan pengguna.

4.2.1.1 Inisialisasi Objek PublicRoomChat

```
public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                  new PublicRoomChat();
            }
      });
}
```

4.2.1.2 Implementasi Pengelolaan PrivateRoomChat

```
// to set a new PrivateRoomChat which will handle a private
messaging.
private     void     setPrivateRoomChatTable(String     _information,
PrivateRoomChat _PriRoCha){
      PrivateRoomChatTable.put(_information, _PriRoCha);
}
```

```
private void addPrivateRoomChat(String _information){
      PriRoCha = new PrivateRoomChat(server, id, name, _information);
// server, targetname.
      setPrivateRoomChatTable(_information, PriRoCha);
}

// to remove a PrivateRoomChat, what it means? it means the private
messaging with a specified users is ended.
private void removePrivateRoomChat(String _information){
      if(PrivateRoomChatTable.containsKey(_information)){
            PrivateRoomChat     PriRoCha     =     (PrivateRoomChat)
PrivateRoomChatTable.get(_information); // server, targetname.
            JOptionPane.showMessageDialog(null, _information + " left
the room.\n This session is going to be expired", " Expired session",
JOptionPane.INFORMATION_MESSAGE);
            PriRoCha.endSession();
            PrivateRoomChatTable.remove(_information);
      }
}
```

4.2.1.3 Implementasi Routing Private Messages

```
public void routeMessage(String _information, String _message){
      if(!PrivateRoomChatTable.containsKey(_information)){
            addPrivateRoomChat(_information);
      }
      PrivateRoomChat       PriRoCha       =       (PrivateRoomChat)
PrivateRoomChatTable.get(_information);
      PriRoCha.addMessage(_message);
      System.out.println("routing:" + _information + ":" + _message);
}
```

4.2.1.4 Implementasi Manajemen Online Client

```
private void rearrangeUsers(){
      String[] allUsers = users.getItems();
      String tmpUser = new String();

      for(int i = 0; i < (allUsers.length - 1); ++i){
            for(int j = i; j < allUsers.length; ++j){
                  if(allUsers[i].compareTo(allUsers[j]) == 1){
                        tmpUser = allUsers[i];
                        allUsers[i] = allUsers[j];
                        allUsers[j] = tmpUser;

                        --i;
                        break;
                  }
            }
```

```
	}

	users.removeAll();

	for(int i = 0; i < allUsers.length; ++i){
		users.add(allUsers[i]);
	}

	if(users.getItemCount() < 1){
		disableUsers();
	}else{
		enableUsers();
	}

	setTotalUser(users.getItemCount() + 1); // including me.
}

public boolean addUser(String _id, String _name){
	String s = new String(_id + " " + _name);
	String[] allUsers = users.getItems();
	boolean isExist = false;
	int i = 0;

	for(i = 0; i < users.getItemCount(); ++i){
		System.out.println("sasas");
		System.out.println(allUsers[i]);
		if(s.equals(allUsers[i])){
			isExist = true;
		}
	}

	if(isExist == false){
		users.add(s);
		rearrangeUsers();
		addMessage("  ***  [" + s + "] joined the room  ***");
	}

	return (isExist);
}
```

```
public void removeUser(String _id, String _name){
      users.remove(_id + " " + _name);
      removePrivateRoomChat(_id + " " + _name);
      rearrangeUsers();
      addMessage("  ***  [" + _id + " " + _name + "] left the room
***");
}
```

4.2.1.5 Implementasi Penambahan Pesan Masuk

```
public void addMessage(String _message){
      jtaMessages.append(_message + "\n");
}
```

4.2.1.6 Implementasi Pengiriman Pesan ke Publik

```
private void processInOTOMessage() {
  String name = inMsgParseToName(inMsg);
  String message = inMsgParseToMessage(inMsg);

  if (!hsOneToOne.containsKey(name)){
    hsOneToOne.put(name, new UIOneToOne(name, this));
  }
  hsOneToOne.get(name).appendMsg("<" + name + ">" + message);
}
```

4.2.2 Implementasi PrivateRoomChat

4.2.2.1 Inisialisasi

```
public PrivateRoomChat(ServerConnection _server, String _id, String
_name, String _information){
  server = _server;
  setID(_id);
  setName(_name);

  StringTokenizer st = new StringTokenizer(_information);
  setOtherID(st.nextToken());
  String _otherName = st.nextToken();
  while(st.hasMoreTokens()){
    _otherName = _otherName + " " + st.nextToken();
  }

  setOtherName(_otherName);
  new Thread(this).start();
}

public void run(){
  initComponents();
}

private void initComponents(){
  mainFrame = new JFrame("  Private Room Chat [user : " + name + "]
with [" + otherName + "]");
  mainPanel = new JPanel();
  mainPanel.setLayout(null);
```

```
// other codes…
```

4.2.2.2 Implementasi Penambahan Pesan Masuk

```
public void addMessage(String _message){
  if(!mainFrame.isVisible()){
    show();
  }
  jtaMessages.append(_message + "\n");
}
```

4.2.3 Implementasi ServerConnection

4.2.3.1 Inisialisasi

```
public ServerConnection(PublicRoomChat _PRC, String _name, String
_serverIP, int _serverPort){
  PRC = _PRC;
  name = _name;

  serverIP = _serverIP;
  serverPort = _serverPort;

  System.out.println(" Trying to connect with : " + serverIP + " " +
serverPort);

  if(setConnection()){
    System.out.println("Connection was established successfully.");
    PRC.connectionEstablished();
    new Thread(this).start();
    isConnected = true;
  }else{
    System.out.println("Connection was refused.");
    PRC.connectionRefused();
    isConnected = false;
  }
}

private boolean setConnection(){
  try{
    socket = new Socket(serverIP, serverPort);
    in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(socket.getOutputStream(), true);
    registerName();
    return (true);
  }catch(ConnectException ce){
    System.out.println("\n A ConnectException occured! (" + ce +
")");
    return (false);
  }catch(IOException ioe){
    System.out.println("\n An IOException occured! (" + ioe + ")");
    return (false);
  }catch(Exception e){
    System.out.println("\n An Exception occured! (" + e + ")");
    return (false);
  }
}
```

```
public void registerName(){
  write("REGISTER " + name);
}
```

4.2.3.2 Implementasi Penerimaan Pesan dan Perintah dari Server

```
private String read(){
  try{
    return(in.readLine());
  }catch(IOException ioe){
    System.out.println("\n An IOException occured! (" + ioe + ")");
    if(isConnected){
      PRC.connectionRefused();
    }
  }catch(Exception e){
    System.out.println("\n An Exception occured! (" + e + ")");
    if(isConnected){
      PRC.connectionRefused();
    }
  }
  return (null);
}

public void write(String _message){
  try{
    out.println(_message);
    System.out.println("write:" + _message);
  }catch(Exception e){
    System.out.println("\n An Exception occured! (" + e + ")");
  }
}

private void tellOtherUser(){
  write("ADD " + id + " " + name);
}

private void tellUser(String _id){
  write("TELL " + _id + " ADD " + id + " " + name);
}

private boolean addUser(String _id, String _name){
  users.remove(_id);
  users.put(_id, _name);

  return(PRC.addUser(_id, _name));
}

private void removeUser(String _id, String _name){
  PRC.removeUser(_id, _name);
}

public void publicChat(String _message){
  write("CHAT PUBLIC " + id + " " + name + ">>" + _message);
  PRC.addMessage(id + " " + name + ">>" + _message);
}

public void privateChat(String _otherID, String _otherName, String
_message){
  write("CHAT PRIVATE " + _otherID + " " + id + " " + name + ">>" +
_message);
```

```
  PRC.routeMessage(_otherID + " " + _otherName, id + " " + name +
">>" + _message);
}

// bellow are needed attributes and methods while this object is
running.

private static final int ID = 0;
private static final int ADD = 1;
private static final int REMOVE = 2;
private static final int CHAT = 3;
private static final int PUBLIC = 4;
private static final int DISCONNECT = 5;
private static final int KICKED = 6;
private static Hashtable keys = new Hashtable();
private static String keystring[] = {"ID", "ADD", "REMOVE", "CHAT",
"PUBLIC", "DISCONNECT", "KICKED"};
static {
  for(int i = 0; i < keystring.length; ++i){
    keys.put(keystring[i], new Integer(i));
  }
}

private int lookup(String _s){
  Integer i = (Integer) keys.get(_s);
  return (i == null ? -1 : i.intValue());
}

// bellow are overiden methods

public void run(){
  String s = new String();
  String keyword = new String("");
  StringTokenizer st;

  while((s = read()) != null){
    System.out.println("SC:" + s);
    st = new StringTokenizer(s);
    keyword = st.nextToken();
    switch(lookup(keyword)){
      case ID:{
        setID(st.nextToken());
        PRC.addMessage("    ***    Welcome  to  the  Public  Room  Chat
***\n");
        tellOtherUser();
        break;
      }
      case ADD:{
        String _id = st.nextToken();
        String _name = st.nextToken();
        while(st.hasMoreTokens()){
          _name = _name + " " + st.nextToken();
        }

        if(!addUser(_id, _name)){
          tellUser(_id);
        }
        break;
      }
      case REMOVE:{
        String _id = st.nextToken();
```

```
        String _name = st.nextToken();
        while(st.hasMoreTokens()){
          _name = _name + " " + st.nextToken();
        }
        removeUser(_id, _name);
        break;
      }
      case CHAT:{
        keyword = st.nextToken();
        if(lookup(keyword) == PUBLIC){
          String _id = st.nextToken();
          String _message = _id.toString();
          while(st.hasMoreElements()){
            _message = _message + " " + st.nextToken();
          }
          PRC.addMessage(_message);
        }else{
          String   _information   =   st.nextToken()   +   "   "   +
st.nextToken();
          String _message = st.nextToken();
          while(st.hasMoreElements()){
            _message = _message + " " + st.nextToken();
          }
          PRC.routeMessage(_information, _message);
          System.out.println("here:"   +   _information   +   ":"   +
_message);
        }
        break;
      }
      case DISCONNECT:{
        PRC.connectionRefused();
        break;
      }
      case KICKED:{
        PRC.kicked();
        break;
      }
      default:{
        System.out.println(s);
      }
    }
  }
}
```