BAB 9
PEMROGRAMAN SOCKET

9.1 Pendahuluan

Seiring dengan pesatnya perkembangan networking di dunia ternyata juga sangat memberi dampak positif yang besar terhadap industri-industri pembuat game seperti yang kita rasakan saat ini dimana banyak game online yang bermunculan di internet diantarannya seperti xian, ragnarok, warcraf dan masih banyak lagi. Bagi para pembuat, pengembang game ataupun yang tertarik untuk membuat game, tentunya sangat menginginkan game yang dihasilkan tidak hanya berjalan secara stand alone tetapi juga di dalam jaringan (multi player). Dalam hal ini, Java dapat menjawab dan memberi solusi kepada para developer untuk membuat game baik di jaringan local maupun global dengan menyertakan pemrograman socket agar permasalahan-permasalahan tersebut dapat teratasi.

9.2 Tujuan:

Setelah mempelajari modul ini peserta diharapkan dapat:

- Memahami konsep jaringan menggunakan socket.
- Membuat program client-server sederhana.
- Memahami dan mampu mengolah data disisi client maupun server

9.3 Socket

Socket adalah salah satu cara untuk komunikasi antar komputer, umumnya lewat network atau internet. Socket biasa digunakan untuk

pemrograman berbasis client-server yang dapat menggunakan socket TCP/IP atau socket UDP

9.3.1 Socket Client

Socket adalah sebuah penghubung antara dua host, yang dapat dibangun dengan tujuh dasar operasi yaitu:

- ✓ Menghubungkan untuk mengendalikan mesin atau perangkat
- ✓ Mengirim data
- ✓ Menerima data
- ✓ Menutup koneksi
- ✓ Bergantung pada port
- ✓ Mendengarkan data yang masuk
- ✓ menerima koneksi dari mesin yang dikendalikan berdasarkan sebuah port

Class Socket pada Java,digunakan oleh client dan server yang mempunyai method yang menangani keempat operasi diatas.Tiga operasi berikutnya hanya dibutuhkan untuk server yang menunggu client untuk koneksi ke server cukup dengan mengimplementasikan class ServerSocket.

Data yang dikirimkan melalui internet di dalam sebuah paket disebut datagra. Setiap datagram berisi header dan payload. header berisi alamat dan port untuk setiap paket, addres dan port untuk tiap paket yang masuk, sedangkan payload berisi data, sejak datagram mempunyai batasan jarak, memungkinkan sebuah paket hilang atau corupted ketika dikirimkan dan membutuhkan pegiriman lagi, untungnya socket memberikan kesempatan kepada programer untuk melakukan koneksi ke jaringan, sebagai contoh

stream di dalam byte dapat dituliskan dan dapat dibaca. Socket melindungi programer dari low-level yang detail dari sebuah jaringan, seperti deteksi kesalahan (error detection), ukuran paket, pengiriman kembali sebuah paket, alamat jaringan dan masih banyak lagi.

Java.net.Socket adalah JFC yang digunakan untuk melakukan operasi TCP client-side. Untuk mendefinisikan host dapat menggunakan InetAddress atau String sedangkan untuk definisi port menggunakan int mulai dari 0 sampai 65.535. Definisi constructornya dapat dilihat sperti dibawah ini:

```
try {
        Socket koneksi = new Socket("192.168.12.138", 80);
  } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
  } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
  }
```

Jika host tidak ditemukan atau tidak berfungsi maka konstructor akan melemparkan `UnknownHostException` tetapi jika socket tidak dapat dibuka maka akan dilemparkan ke `IOException` ada beberapa hal yang menyebabkan permintaan koneksi gagal misalnya host tujuan mungkin tidak menerima koneksi, koneksi dial-up terputus atau permasalahan routing menolak host tujuan(rejected).Selain metode diatas definisi socket dapat dilakukan dengan menggunakan class InetAddress.

9.3.2 Socket Server

Socket Server digunakan untuk menangani seluruh request dari jaringan yang disebut dengan socket client. Berikut ini adalah cara pendefinisian konstruktor dari class ServerSocket yang akan digunakan untuk menangani koneksi dari Client.

```
try {
      ServerSocket s=new ServerSocket(90);
      s.accept();
} catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
}
```

Konstruktor dari ServerSocket diatas akan membuka atau menerima koneksi melalui port 90, kemudian method accept() yang dipanggil untuk menunggu koneksi dari client. Konstruktor akan melemparkan exception apabila port yang didefinisikan diatas telah dibuaat sebelumnya. Potongan kode berikut digunakan untuk memeriksa port yang telah digunakan(terbuka).

```
for(int i=0;i<1024;i++){
   try {
        ServerSocket s=new ServerSocket(i);
   }catch (IOException e) {
        System.out.println("Port ini telah digunakan : "+i);
   }
}
```

Selain metode diatas class ServerSocket juga menyediakan pilihan untuk membatasi jumlah koneksi dari client menggunakan dua parameter dimana parameter pertama adalah nomor port dan parameter kedua adalah jumlah koneksi yang akan diterima, potongan kodenya dapat dilihat sebagai berikut.

```
try {
        ServerSocket s=new ServerSocket(90,10);
}catch (IOException e) {

}
```

Kebanyakan server yang dipakai sekarang menggunakan lebih dari satu IP Address untuk itu class ServerSocket juga menyediakan konstruktor yang digunakan untuk memilih IP mana yang akan digunakan, cotohnya sebagai berikut:

```
try {
      ServerSocket s=new ServerSocket(100,10,
InetAddress.getByName("192.168.12.138"));

} catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
}
```

Konstruktor diatas melewatkan tiga parameter yang pertama adalah nomor port kemudian parameter, kedua adalah banyaknya koneksi yang akan diterima dan yang terakhir yaitu ip addres yang akan menerima permintaan koneksi dari client dimana parameter ketiga ini menggunakan class InetAddress untuk melewatkan IP Address yang akan digunakan.

3.4 Membuat Server Sederhana

Setelah memahami tentang konstruktor dari ServerSocket yang terdapat pada bab II, sekarang kita akan membuat server sederhana yang akan digunakan untuk menangani koneksi dari client. Berikut ini adalah kode program dari class Server.

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Server implements Runnable
{
    private ServerSocket server;

    public Server(int port)
    {
        try
        {
            server = new ServerSocket(port);
        }
        catch(IOException e)
        {
            System.out.println("Port : "+port+ " telah digunakan");
            System.exit(1);
        }
        System.out.println("-> Server berhasil dibuat dengan port
"+port);
    }
```

```
      public void run()
    {
        while(true)
        {
            try
            {
                System.out.println("Menunggu konesi dari
client...");
                new ProsesClient(server.accept());
            }
            catch(IOException e){}
        }
    }
    public static void main(String args[])
    {
        Server server = new Server(5000);
        new Thread(server).start();
    }
}
```

Konstruktor dari class Server diatas melewatkan parameter berupa nilai integer yang akan dugunakan sebagai nomor port dari server. Jika port yang kita masukan telah digunakan maka program akan melemparkan exception. Kemudian program kita akan menunggu koneksi dari client dengan menggunakan bantuan dari thread yang akan kita bahas nanti. Method server.accept() dari object ServerSocket yang kita buat akan menunggu koneksi yang kemudian dilewatkan lagi atau ditangani pada konstruktor dari class ProsesClient.

Ada beberapa peraturan yang harus kita gunakan untuk menangani data data yang diterima oleh server agar program dapat berjalan sesuai dengan keinginan. Untuk itu kita harus membuat interface Protocol yang akan kita implementasikan pada class yang mmbutuhkan peraturan-peraturan yang diinialisasikan pada interface Protocol tersebut. Berikut ini adalah listing programnya.

```
public interface Protocol {
    public static final int CHAT_USER_BARU = 0;
    public static final int CHAT_HAPUS_USER = 1;
    public static final int CHAT_PESAN_USER = 2;
    public static final int SET_USER = 3;
}
```

Interface Protocol diatas menetapkan beberapa peraturan yang bertype integer dan nilainya tidak dapat diubah ataupun dioverride. Ada beberapa variable diatas antara lain CHAT_USER_BARU dalam program yang akan kita buat nanti digunakan untuk pengecekan user yang yang baru terkoneksi dan datanya akan dikirimkan ke seluruh client yang online dalam hal ini data yang dikirimkan berupa nama user(“nick ???? telah bergabung di room ini”), CHAT_HAPUS_USER digunakan sebagai validasi untuk memberitahu kepada seluruh client yang terkoneksi bahwa user telah putus koneksinya dari server (“ nick ???? keluar dari room”), CHAT_PESAN_USER digunakan sebagai validasi dimana datanya juga dikirimkan keseluruh client yang aktif dan akan ditampilkan berupa pesan, SET_USER diginakan untuk memberikan nama dari client disisi server (setiap object yang dibuat oleh class server melalui class ProsesClient).

Semua penjelasan tentang peraturan-peraturan yang dibuat pada inteface Protocol akan diterapkan pada class ProsesClient yang potongan kode programnya adalah sebagai berikut.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ProsesClient implements Protocol{
    Socket client;
    boolean koneksi;
    private TerimaData terimadata;
    private KirimData kirimdata;
    public static ArrayList clientList = new ArrayList();
    String nama;
```

Class ProsesClient diatas mengimplementasi Protocol yang akan digunakan sebagai rule dari data yang akan diproses, kemudian ada beberapa variable yaitu antara lain *client* yang digunakan sebagai referensi dari socket

yang diterima oleh server melalui socket.accept() yeng dilewatkan melalui konstruktor ProcesClient, *koneksi* bertype boolean dimana akan di set true apabila client berhasil terkoneksi, *terimadata* adalah inner class yang digunakan untuk menghandle data yang masuk dari server, *kirimdata* juga berupa inner class yang digunakan untuk mengirim data, *clientList* digunakan untuk menambpung setiap client yang terkoneksi ke server, *nama* digunakan sebagai nama dari setiap client yang terkoneksi.

```
ProsesClient(Socket client){
  try{
      this.client = client;

      DataInputStream in = new DataInputStream
(client.getInputStream());
      DataOutputStream out = new DataOutputStream
(client.getOutputStream());

      terimadata = new TerimaData(in);
      kirimdata = new KirimData(out);

      koneksi = true;

      synchronized(clientList)
      {
          clientList.add(this);
      }

   }catch(IOException e){
            System.out.println("Tidak dapat terkoneksi: "+e);
   }
}
```

Program diatas merupakan konstruktor untama dari class ProsesClient yang melewatkan data berupa object Socket. Didalam konstruktor tersebut juga ada DataInputStream yang akan digunakan untuk menerima data dan DataOuputStream untuk mengirimkan data dimana kedua-duanya di lewatkan melalui konstruktor dari inner class TerimaData(in) dan KirimData(out). Apabila terjadi kesalahan maka akan melemparkan exception. Selanjutnya adalah inner class TerimaData yang potongan kodenya seperti dibawah ini.

```
  public class TerimaData implements Runnable
    {
```

```
    Thread loop;
      private DataInputStream in;
      public TerimaData(DataInputStream in)
      {
          this.in = in;
          loop = new Thread(this);
          loop.start();
      }

      public void run()
      {
          Thread thisThread = Thread.currentThread();
          while(loop==thisThread)
          {
              try
              {
                  String data = in.readUTF();
                  prosesData(data);
              }

          catch(IOException e)
              {
                  disconnect();
              }
          }
      }

      public void destroy()
      {
          loop = null;
      }
  }
```

Inner class TerimaData diatas melewatkan object DataInputStream dari konstruktornya dan juga dithread untuk membaca data yang dikirimkan oleh client dengan in.readUTF() yang kemudian diolah melalui method prosesData(). Jika terjadi kesalahan maka akan memanggil method disconnect(). Method prosesData() dibawah ini mempunyai satu parameter bertype String dimana parameter ini adalah data yang ditangkap dari client yang kemudian diproses dengan bantuan dari StringTokenizer yang digunakan untuk memecah data dengan pemisah " | " kemudian datanya yang pertama dikonversi menjadi Integer dan kemudian diolah berdasarkan peraturan-peraturan yang kita buat pada interface Protocol diatas.

```
  public void prosesData(String data){
      StringTokenizer st = new StringTokenizer(data, "|");
    int kode = Integer.parseInt(st.nextToken());
```

```
        switch(kode)
        {
                case SET_USER:{
                        nama=st.nextToken();
                        break;
                }
                                case CHAT_USER_BARU :{
                        broadcastToClient(data);
                        break;
                }
                case CHAT_HAPUS_USER :{
                        broadcast(data);
                        break;
                }
                case CHAT_PESAN_USER :{
                        broadcast(data);
                        break;
                }
        }
    }
```

Seperti yang terlihat pada program diatas terdapat method broadcastToClint() yang brfungsi untuk mengirimkan data keseluruh client kecuali client yang mengirim data sedangkan broadcast() mengirimkan data keseluruh client termasuk client yang mengirimkan data.

```
public static void broadcast(String data)
    {
        synchronized(clientList)
        {
            ProsesClient user;
            for(int i=0; i<clientList.size(); i++)
            {
                user = (ProsesClient)clientList.get(i);
                user.kirimData(data);
            }
        }
    }
    public void broadcastToClient(String data)
    {
        synchronized(clientList)
        {
            ProsesClient user;

            for(int i=0; i<clientList.size(); i++)
            {
                user = (ProsesClient)clientList.get(i);
                if(user!=this)
                    user.kirimData(data);
            }
        }
    }
```

Sekarang kita akan membahas tentang pengiriman data dari server dengan inner class KirimData seperti dibawah ini.

```
public class KirimData implements Runnable {
      Thread loop;
        LinkedList tampungData;
        DataOutputStream out;
        public KirimData(DataOutputStream out)
        {
            this.out = out;
            tampungData = new LinkedList();
            loop = new Thread(this);
            loop.start();
        }
        public void tambahData(String data)
        {
            synchronized(tampungData)
            {
                  tampungData.add(data);
                  tampungData.notify();
            }
        }

        public void run()
        {
            String data;

            Thread thisThread = Thread.currentThread();
            while(loop==thisThread)
            {
                synchronized(tampungData)
                {
                    if(tampungData.isEmpty() && loop!=null)
                    {
                        try
                        {
                              tampungData.wait();
                        }
                        catch(InterruptedException e) { }
                    }
                }
                while(tampungData.size()>0)
                {
                    synchronized(tampungData)
                    {
                        data = (String)tampungData.removeFirst();
                    }

                    try
                    {
                        out.writeUTF(data);
                    }
                    catch(IOException e)
                    {
                        disconnect();
                    }
                 }
            }
        }
```

```
        public void destroy()
        {
            loop = null;
            synchronized(tampungData)
            {
                tampungData.notify();
            }
        }
}
```

Inner class KirmData diatas menggunakan thread agar pengiriman data dapat dilakukan setiap saat dimana data yang dikirimkan tidak langsung dikirim dengan DataOutputStream tetapi masih di tambahkan kedalam LinkedList sebagai antrian kemudian dikirimkan dengan out.WriteUTF() dimana apabila data di object tampungData kosong maka program akan menunggu dengan menggunakan method tampungData.wait(), dimana method ini akan diinterupsi apabila terjadi penambahan data melalui method tambahData() karena disana method tampungData.notify() dipanggil. Walaupun pada inner class KirimData menyediakan method tambahData() yang melewatkan parameter berupa String tetapi dalam kenyataanya kita tidak langsung mengakses kesana tetapi masih membuat method lagi yaitu

```
public void kirimData(String data)
{
    kirimdata.tambahData(data);
}
```

Kita membuat satu method lagi yaitu disconnect() yang akan digunakan untuk mengirimkan keseluruh client yang aktif yang memberitahukan bahwa koneksi dari client telah terputus. Berikut potongan kode programnya.

```
public synchronized void disconnect(){
        if(koneksi){
                synchronized(clientList){
                clientList.remove(this);
                }

            broadcast(CHAT_HAPUS_USER+"|"+nama);
            koneksi = false;
            kirimdata.destroy();
```

```
            terimadata.destroy();
            try{  client.close();   }
            catch(Exception e) {}
            client = null;
        }
        System.out.println("Client Disconnected");
}
```

3.5 Membuat Client Sederhana

Sebagai lanjutan dari sisi server dimana pusatnya seluruh data diolah, sekarang kita akan membuat aplikasi Client yang melakukan koneksi ke server yang baru saja dibuat tadi. Disini kita hanya membuat dua class dan sebuah interface yaitu class NetworkData, Client dan interface NetworkDataListener. Program yang akan dibangun dititikberatkan pada pengiriman, penerimaan dan pemrosesan data yang di ilustrasikan dalam aplikasi berupa chatting.

Program yang akan kita buat juga akan mengimplementasi interface Protocol yang telah dibuat pada pembahasan server diatas. Sekarang mari kita mulai dengan membuat interface NetworkDataListener sebagai berikut.

```
public interface NetworkDataListener {
      public void menerimaData(String data);
      public void networkDisconnected(String data);
}
```

Intrface NetworkDataListener akan diimplementasikan pada class utama yaitu Client dimana fungsinya untuk menerima data yang dikirimkan oleh server. NetworkDataListener akan kita gunakan sebagai parameter dari class NetworkData seperti berikut ini.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class NetworkData implements Protocol{
      NetworkDataListener NDL;
      Socket client;
      boolean koneksi;
      private TerimaData terimadata;
      private KirimData kirimdata;
      NetworkData(String host,int port,NetworkDataListener NDL){
```

```
        try {
          this.NDL=NDL;
          client=new Socket(host,port);
          DataInputStream in = new
DataInputStream(client.getInputStream());
          DataOutputStream out = new
DataOutputStream(client.getOutputStream());
          terimadata = new TerimaData(in);
          kirimdata = new KirimData(out);
          koneksi = true;
        } catch (UnknownHostException e) {
             e.printStackTrace();
        } catch (IOException e) {
             e.printStackTrace();
        }
}
```

Class NetworkData akan mengimplementasi interface Protocol sebagai aturan yang diberlakukan untuk metode pengiriman data, adapun konstruktornya melewatkan tiga parameter yaitu host, port dan NetworkDataListener, didalam class ini terdapat dua inner class yang isinya tidak jauh berbeda dengan inner class yeng berada di sisi server pada bahasan di atas. Apabila koneksi gagal dilakukan maka akan melemparkan exception. Berikut ini adalah program inner class TerimaData.

```
public class TerimaData implements Runnable{
       Thread loop;
       private DataInputStream in;
       public TerimaData(DataInputStream in)
       {
            this.in = in;
            loop = new Thread(this);
            loop.start();
       }

       public void run()
       {
            Thread thisThread = Thread.currentThread();
            while(loop==thisThread)
            {
               try
               {
                    String data = in.readUTF();
                    NDL.menerimaData(data);
               }
               catch(IOException e)
               {
                    disconnect();
               }
             }
         }
```

```
        public void destroy()
        {
                loop = null;
        }
}
```

Seperti yang terlihat object bernama NDL yang dibuat dari NetworkDataListener digunakan ketika data diterima menggunakan in.readUTF() dan kemudian memanggil dan melewatkan data yang dibaca tadi pada method yang ada didalam interface tersebut yaitu NDL.menerimaData(...). Perlu diketahui bahwa semua program yang mengimplementasikan interface NetworkDataListener akan secara otomatis memanggil method menerimaData(...) maka disinilah kesempatan kita untuk memproses data. Untuk inner class KirimData tidak ada perbedaan dengan KirimData yang ada di server. Berikut adalah listing programnnya.

```
public class KirimData implements Runnable{
      Thread loop;
      LinkedList tampungData;
      DataOutputStream out;
      public KirimData(DataOutputStream out){
            this.out = out;
            tampungData = new LinkedList();
            loop = new Thread(this);
            loop.start();
      }
      public void tambahData(String data){
            synchronized(tampungData)
            {
                  tampungData.add(data);
                  tampungData.notify();
            }
      }
            public void run()
            {
                  String data;
                  Thread thisThread = Thread.currentThread();
                  while(loop==thisThread)
                  {
                        synchronized(tampungData)
                        {
                              if(tampungData.isEmpty() && loop!=null)
                              {
                                    try{
                                          tampungData.wait();
                                    }catch(InterruptedException e) { }
                              }
```

```
                        }
                        while(tampungData.size()>0)
                        {
                            synchronized(tampungData)
                            {
                                data =
(String)tampungData.removeFirst();
                            }
                            try{
                                out.writeUTF(data);
                            }
                            catch(IOException e){
                                disconnect();
                            }
                        }
                    }
                }
                public void destroy()
                {
                    loop = null;
                    synchronized(tampungData)
                    {
                          tampungData.notify();
                    }
                }
}
```

Method disconnect juga sama dengan yang ada di server, potongan kodenya adalah sebagai berikut.

```
public synchronized void disconnect(){
        if(koneksi)
        {
            NDL.networkDisconnected("disconnected..");

            koneksi = false;

            kirimdata.destroy();
            terimadata.destroy();

            try
            {
                client.close();
            }
            catch(Exception e) {}
            client = null;
        }

        System.out.println("Client Disconnected");
}
public void kirimData(String data){
        kirimdata.tambahData(data);
}
```

Method disconnect diatas juga akan memanggil NDL.network Disconnected(...) dimana seluruh class yang mengimplementasi

NetworkDataListener akan memanggil secara otomatis method networkDisconnected(...).

Sekarang membuat satu class lagi yaitu class Client yang akan digunakan sebagai GUI dan sekaligus akhir dari program chating yang akan kita bangun. Program ini menggunakan dua panel yaitu panel pertama menampung nama, host dan port dan tombol connect kemudian panel yang kedua yaitu isi, pesan dan tombol send. Pertama kali dijalanan adalah panel pertama jika tombol connect ditekan maka panel pertama akan dihapus dan digantikan dengan panel kedua sebagai tampilan GUInya. Program ini mengimplementasi NetworkDataListener, Protocol dan ActionListener. Layout dari container diset null oleh sebab itu semua letak dari componen menggunakan bentuk koordinat dengan menggunakan method reshape(posisi x, posisi y, panjang, tinggi) . Berikut ini adalah listing program dari class Client.

```
public class Client extends JFrame implements
NetworkDataListener,Protocol, ActionListener {

	public static void main(String[] args) {
		new Client();
	}

	private String host;
	private String nama;
	private int port;
	NetworkDataListener NDL=this;
	NetworkData ND;

	JPanel p1=new JPanel();
	JLabel l1=new JLabel("Nama :");
	JLabel l2=new JLabel("Host :");
	JLabel l3=new JLabel("Port :");
	JTextField NAMA=new JTextField("anonymouse");
	JTextField HOST=new JTextField("127.0.0.1");
	JTextField PORT=new JTextField("5000");
	JButton connect=new JButton("Connect");


	JPanel p2=new JPanel();
	JButton send=new JButton("SEND");
	JTextArea pesan=new JTextArea(15,2);
	JTextArea isi=new JTextArea(2,2);

	Container c;
```

```
    Client(){
        c=getContentPane();
        c.setLayout(null);
        p1.setLayout(null);
        p1.add(l1).reshape(0, 0, 100, 25);
        p1.add(l2).reshape(0, 40, 100, 25);
        p1.add(l3).reshape(0, 80, 100, 25);
        p1.add(NAMA).reshape(50, 0, 100, 25);
        p1.add(HOST).reshape(50, 40, 100, 25);
        p1.add(PORT).reshape(50, 80, 100, 25);
        p1.add(connect).reshape(30, 120, 100, 25);


        p2.setLayout(null);
        isi.setEditable(false);
        p2.add(new JScrollPane(isi)).reshape(0, 0, 470, 200);
        p2.add(new JScrollPane(pesan)).reshape(0, 220, 350, 55);
        p2.add(send).reshape(370, 220, 100, 25);

        c.add(p1).reshape(150, 50, 200, 200);


        connect.addActionListener(this);
        send.addActionListener(this);

        setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        setSize(500,500);
        setVisible(true);
        setResizable(false);
    }
```

Jika tombol connect ditekan maka instance dari NetworkData yaitu ND akan dibuat dimana akan menangani pesan yang diterima ataupun yang dikirimkan. Selain itu juga mengirimkan data keserver untuk memberi nama pada object ProsesClient yang menangani data yang masuk atau keluar dengan menggunakan ND.kirimData (SET_USER+"|"+nama), juga mengirimkan ND.kirimData (CHAT_USER_BARU+"|"+ nama), dimana jika data ini diterima oleh server maka akan langsung mengirimkan ke seluruh client yang aktif bahwa user telah bergabung ke room.

```
public void actionPerformed(ActionEvent e) {

        if(e.getSource() == connect){
            this.host=HOST.getText();
            this.port=Integer.parseInt(PORT.getText());
            this.nama=NAMA.getText();

            ND=new NetworkData(host,port,NDL);
            ND.kirimData(SET_USER+"|"+nama);
```

```
                    ND.kirimData(CHAT_USER_BARU+"|"+nama);
                    c.remove(p1);
                    c.add(p2).reshape(10, 10, 500, 500);
                    setTitle(nama);
                    this.validate();
              }
              if(e.getSource()== send){
                    ND.kirimData(CHAT_PESAN_USER+"|"+nama+" :
pesan.getText());
              }
 }
```

Ketika tombol send ditekan maka program akan mengirimkan data melalui method ND.kirimData(CHAT_PESAN_USER+"|"+nama" : "+pesan.getText()); dimana ketika data sampai diserver maka akan di periksa menggunakan Protocol yang telah ditetapkan dan kemudian mengirimkan data ke seluruh client.

Method menerimaData(...) yang dioverride dari interface NetworkDataListener akan digunakan untuk memeriksa data yang diterima oleh client, dimana data-data tersebut akan diproses sesuai dengan Protocol yang telah ditetapkan, pengolahan datanya juga menggunakan StringTokennizer dengan tanda pemisah adalah " | " sama seperti yang dilakukan pada sisi server. Apabila ternyata pesannya berupa CHAT_USER_BARU maka object TextArea(isi) akan ditambahkan isi.append("nick : "+nick+" telah bergabung di channel ini\n"); jika ternyata pesannya berupa CHAR_HAPUS_USER maka yang akan ditambahkan kedalam object TextArea(isi) isi.append("nick : "+nick+" telah keluar dari channel ini!\n"); tetapi apabila pesannya berupa CHAT_PESAN_USER maka object TextArea(isi) akan ditambahkan isi.append(st.nextToken()+"\n"). Apabila terjadi disconnect disisi client maka class NetworkData yang ada disisi client akan mengirimkan pesan "disconnect".