Tutorial 1

Pemrograman Socket

Tujuan:
- Mahasiswa mampu membuat program untuk menangani nama dan IP komputer
- Mahasiswa mampu membuat program komunikasi antar proses dengan soket
- Mahasiswa mampu membuat program client untuk layanan server yang ada
- Mahasiswa mampu membuat program untuk mengirim obyek lewat soket.

Universitas Atma Jaya Yogyakarta
Fakultas Teknologi Industri
Teknik Informatika
Kusnadi@mail.uajy.ac.id

A. Menangani Nama dan IP Komputer dengan Java

Sistem terdistribusi terdiri atas kumpulan komputer yang berdiri sendiri (otonom) dan saling bekerja sama, dan mungkin juga berbagi pakai sumber daya komputasi (prosesor, memori, storage) untuk menyediakan layanan bagi pengguna. Untuk dapat berkomunikasi dengan komputer lain, masing-masing komputer diberi identitas pada level aplikasi berupa alamat IP (Internet Protocol) seperti 192.168.29.251.

Namun mengenali alamat komputer berdasarkan penomeran IP diatas bukanlah hal yang mudah dilakukan dengan ingatan manusia (programmer!). Oleh sebab itu pada infrastruktur jaringan terdapat layanan *Naming*, seperti DNS (Domain Naming Service) pada internet atau intranet yang menyimpan semacam kamus translasi nama *user friendly* komputer dan alamat IP nya. Tugas layanan naming semacam ini menerjemahkan alamat komputer yang *user-friendly*, seperti *www.google.com* menjadi alamat IP seperti 66.102.7.104.

Pada komputer lokal juga terdapat layanan semacam ini yang disebut dengan *resolver*, yang menyimpan kamus translasi dalam suatu file khusus (di Windows XP file tersebut biasanya terletak di *C:\Windows\system32\drivers\etc\hosts*). Contoh isi file hosts dapat dilihat pada gambar 1. Jadi jika komputer lokal diminta menghubungi suatu nama komputer, maka komputer lokal akan berusaha menerjemahkan nama komputer tersebut ke alamat IP sesungguhnya dengan menggunakan resolver, jika entry nama komputer tidak terdapat pada file *hosts* maka komputer lokal kemudian akan mencoba menghubungi layanan DNS (berdasarkan setting DNS Server pada komputer).

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

127.0.0.1                   localhost
192.168.29.21               jarkom-01
```

Gambar 1. Contoh isi file hosts

Java menyediakan obyek *InetAddress* yang memanfaatkan layanan sistem operasi dan jaringan diatas untuk melakukan translasi nama dan IP komputer. Demo dibawah ini memperlihatkan bagaimana menggunakan obyek InetAddress untuk mengambil IP komputer lokal (D1-1) dan nama komputer lokal (D1-2), melakukan translasi IP ke nama komputer (D1-3), serta translasi nama ke IP komputer (D1-4) (seperti perintah shell NSLookup).

Demo D1-1

1. Buat program di bawah ini, simpan dengan nama **getIP.java**

```
import java.net.*;

public class getIP {
  public static void main(String args[]) throws Exception {
    InetAddress host = null;
    host = InetAddress.getLocalHost();
    byte ip[] = host.getAddress();
    for (int i=0; i<ip.length; i++) {
      if (i > 0) {
        System.out.print(".");
      }
      System.out.print(ip[i] & 0xff);
    }
    System.out.println();
  }
}
```

2. Kompilasi program diatas, jalankan dan amati hasilnya

```
$ javac getIP.java
$ java getIP
```

Demo D1-2

1. Buat program di bawah ini, simpan dengan nama **getName.java**

```
import java.net.*;

public class getName {
  public static void main(String args[]) throws Exception {
    InetAddress host = null;
    host = InetAddress.getLocalHost();
    System.out.println("Nama komputer Anda: " +
host.getHostName());
  }
}
```

2. Kompilasi dan jalankan program diatas (lihat D1-1) dan amati hasilnya

```
$ javac getName.java
$ java getName
```

Demo D1-3

1. Buat program di bawah dengan nama **IPtoName.java**

```
import java.net.*;
public class IPtoName {
  public static void main(String args[]) {

    if (args.length == 0) {
      System.out.println("Pemakaian: java IPtoName <IP address>");
      System.exit(0);
    }

    String host = args[0];
    InetAddress address = null;

    try {
      address = InetAddress.getByName(host);
    } catch (UnknownHostException e) {
      System.out.println("invalid IP - malformed IP");
      System.exit(0);
    }

    System.out.println(address.getHostName());
 }
}
```

2. Kompilasi dan jalankan dengan argumen IP komputer lokal dan komputer lain

```
$ javac IptoName.java
$ java IPtoName <IP-address-Anda>
$ java IPtoName <IP-address-teman-Anda>
$ java IptoName <IP-address-sembarang>
```

Demo D1-4

1. Buat program ini, simpan dengan nama **NsLookup.java:**

```
import java.net.*;

public class NsLookup {
  public static void main(String args[]) {

    if (args.length == 0) {
      System.out.println("Pemakaian: java NsLookup <hostname>");
      System.exit(0);
    }
```

```
    String host = args[0];
    InetAddress address = null;

    try {
      address = InetAddress.getByName(host);
    } catch(UnknownHostException e) {
      System.out.println("Unknown host");
      System.exit(0);
    }

    byte[] ip = address.getAddress();
    for (int i=0; i<ip.length; i++) {
      if (i > 0) System.out.print(".");
      System.out.print((ip[i]) & 0xff);
    }

    System.out.println();
  }
}
```

2. Kompilasi dan jalankan dengan cara (coba dengan hostname yang berbeda-beda)

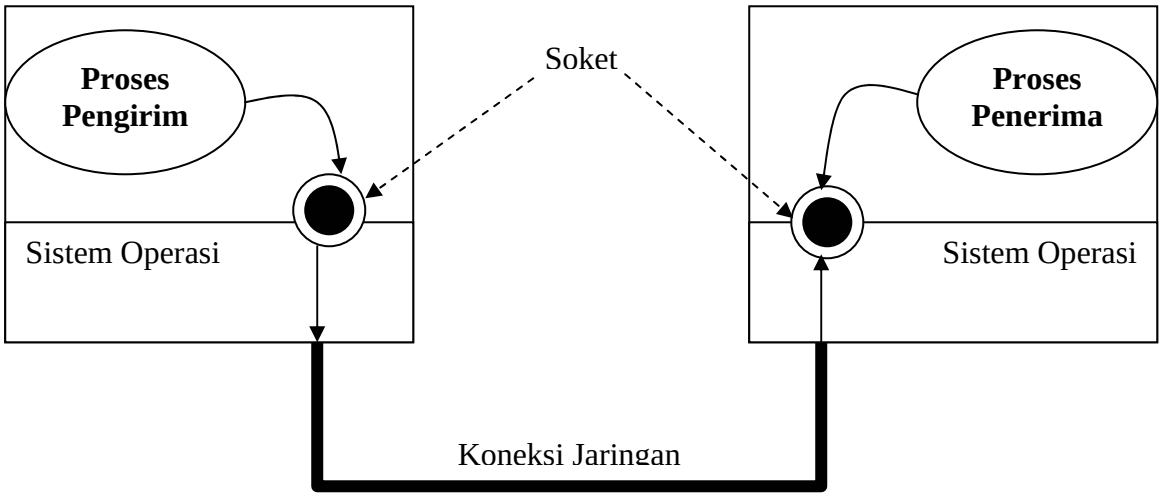
```
$ javac NsLookup.java
$ java NsLookup <hostname>
```

B. Konsep Komunikasi Antar Proses

Dalam sistem yang demikian terjadi komunikasi antar proses-proses yang berada pada computer yang berbeda.

Apa itu Soket?

Soket merupakan fasilitas IPC (Inter Proses Communication) untuk aplikasi jaringan. Model komunikasi dengan soket digambarkan pada gambar 1



Gambar 2. Model IPC dengan soket

Agar suatu soket dapat berkomunikasi dengan soket lainnyanya, maka soket butuh diberi suatu alamat unik sebagai identifikasi. Alamat soket terdiri atas Alamat IP dan Nomer Port. Contoh alamat soket adalah **192.168.29.30**: **3000**, dimana nomer 3000 adalah nomer portnya. Alamat IP dapat menggunakan alamat Jaringan Lokal (LAN) maupun alamat internet. Jadi soket dapat digunakan untuk IPC pada LAN maupun Internet.

Mengapa dibutuhkan nomer port? Apakah nomer IP komputer tujuan saja tidak cukup? Nomer port dibutuhkan karena proses yang berjalan pada suatu komputer umumnya lebih dari satu. Sehingga dibutuhkan tambahan informasi sebagai identifikasi proses yang hendak dihubungi. Jika IP computer diibaratkan adalah nomer telepon suatu perusahaan, maka nomer port adalah nomer ekstensinya. Suatu proses yang hendak berkomunikasi dengan proses lain lewat mekanisme soket haruslah mengikatkan dirinya dengan salah satu port pada komputerny. Pengikatan diri ini disebut dengan *binding*.

Macam-macam Komunikasi Soket

Secara umum ada dua macam komunikasi dengan menggunakan soket, yaitu komunikasi stream dan komunikasi datagram. Komunikasi stream sering juga disebut dengan komunikasi yang berorientasi koneksi (*Connection oriented communication*). Sedangkan Komunikasi datagram disebut juga dengan komunikasi tak berkoneksi (*connectionless communication*). Protokol standar untuk komunikasi stream dikenal denga istilah TCP (*Transmission Control Protocol*), sedangkan standar protokol komunikasi datagram dikenal dengan UDP (*User Datagram Protocol)*.

Pada UDP, setiap kali suatu paket data dikirim, informasi soket pengirim dan alamat soket tujuan turut dikirimkan. Hal demikian tidak dibutuhkan oleh TCP, karena TCP akan membuat setup koneksi dengan soket tujuan terlebih dulu. Setelah koneksi terbentuk, tidak dibutuhkan mengirimkan informasi soket pengirim tiap kali data dikirimkan. Ini karena proses tujuan akan mengidentifikasi setiap data yang tiba pada soket tujuan sebagai data dari proses pengirim. Koneksi yang terbentuk pada TCP bersifat dua arah (*bidirectional*).

Perbedaan lain adalah UDP memiliki batasan ukuran datagram (paket data) yang dikirimkan sebesar 64 kb. Sedangkan TCP tidak memiliki batasan ini karena data-data dikirimkan sebagai aliran data (stream). Sesungguhnya TCP akan memecah data yang besar menjadi sejumlah paket data berukuran kecil dan diberi nomer urut. Pada sisi soket penerima, paket-paket data ini akan disimpan, diurutkan kembali, dan akhirnya digabungkan kembali menjadi data besar.

Perbedaan lain adalah UDP merupakan protocol yang *unreliable* (tidak handal). Ketika paket data dikirimkan, UDP tidak mengecek kembali apakah data yang dikirim sampai tujuan. Jadi dengan UDP tidak ada kepastian bagi sisi pengirim bahwa datanya sudah sampai ke tujuan dengan keadaan baik. Sebaliknya TCP adalah protocol yang *reliable* yang senantiasa menunggu konfirmasi dari pihak soket penerima, dan kalau perlu paket data yang hilang akan dikirimkan kembali. Konsekuensinya adalah TCP menimbulkan overhead lalulintas jaringan lebih tinggi dibanding UDP.

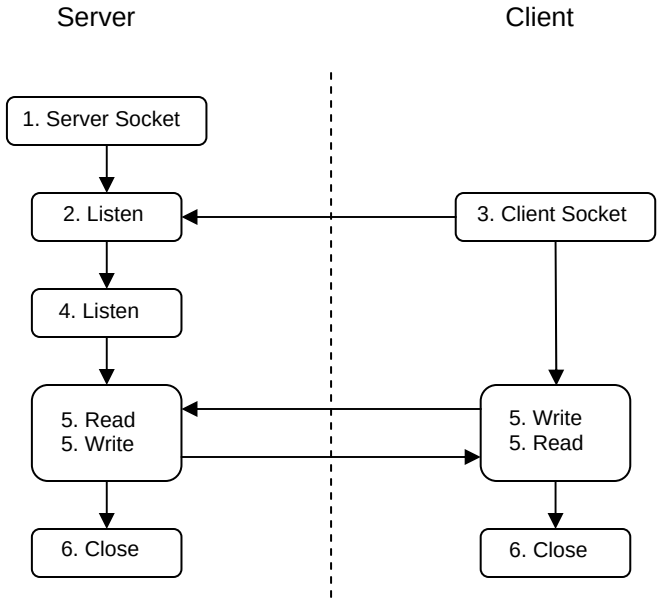
Model Aplikasi Client Server

Model aplikasi yang menggunakan komunikasi soket dengan protokol TCP digambarkan pada gambar 2. Obyek soket pada sisi client dan server berbeda sedikit. Pada

sisi aplikasi server, suatu soket server dibentuk (1) dan melakukan operasi *listen (2)*. Operasi ini pada intinya menunggu permintaan koneksi dari sisi client. Sedangkan pada sisi client, dibentuk suatu soket biasa.

Pada saat soket client (3), informasi alamat soket server dilewatkan sebagai argumen dan soket client akan otomatis mencoba meminta koneksi ke soket server. Pada saat permintaan koneksi client sampai pada server, maka server akan membuat suatu soket biasa. Soket ini yang nantinya akan berkomunikasi dengan soket pada sisi client. Setelah itu soket server dapat kembali melakukan *listen* (4) untuk menunggu permintaan koneksi dari client lainnya. Langkah 4 ini umumnya hanya dilakukan jika aplikasi server mengimplementasikan multithreading.

Setelah tercipta koneksi antara client dan server, maka keduanya dapat saling bertukar pesan (5). Salah satu atau keduanya kemudian dapat mengakhiri komunikasi dengan menutup soket (6).



Gambar 3. Model Aplikasi Clien/Server pada protokol TCP

Untuk protokol UDP, perbedaanya adalah soket di sisi server sama dengan soket di sisi client, dan tidak ada operasi listen pada sisi server. Kemudian saat paket data dikirimkan, alamat soket penerima harus disertakan sebagai argumen.

C. Pemrograman Soket di Java

Java menyediakan obyek ***Socket*** dan ***ServerSocket*** untuk komunikasi soket TCP. ServerSocket digunakan pada sisi aplikasi server, sedangkan Socket digunakan baik pada sisi aplikasi server maupun client. Berikut adalah langkah-langkah membuat komunikasi soket di java:

1. Membuka socket

   Pada client, hal ini dilakukan sebagai berikut:

```
Socket myKlien = null;
try {
  myKlien = new Socket("host", NomorPort);
} catch (UnknownHostException uhe) {
  uhe.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

   Pada server:

```
public static final int NomorPort = 1234;
ServerSocket Layanan = null;
try {
  Layanan = new ServerSocket(NomorPort);
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

   Selain itu, pada server juga harus dibuat sebuah objek socket dari kelas ServerSocket untuk mendengar dan menerima koneksi dari klien, sebagai berikut:

```
Socket layananSocket = null;
try {
  layananSocket = Layanan.accept();
} catch (IOException iex) {
  iex.printStackTrace();
}
```

2. Membuat data input stream

   Untuk membuat input stream pada client, dapat digunakan kelas BufferedReader untuk menerima respon dari server.

```
BufferedReader is = null;
try {
  is              =              new              BufferedReader(new
InputStreamReader(myKlien.getInputStream()));
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

   Pada server juga dapat digunakan BufferedReader untuk menerima input dari client.

```
BufferedReader is = null;
try {
  is              =              new              BufferedReader(new
InputStreamReader(Layanan.getInputStream()));
} catch (IOException ioe) {
  ioe.printStackTrace();
}
```

3. Membuat data output stream

   Pada client, dapat digunakan kelas DataOutputStream untuk mengirimkan data ke socket server.

```
DataOutputStream os = null;
try {
  os = new DataOutputStream(myKlien.getOutputStream());
} catch (IOException ix) {
  ix.printStackTrace();
}
```

   Pada server:

```
DataOutputStream os = null;
try {
  os = new DataOutputStream(Layanan.getOutputStream());
} catch (IOException ie) {
  ie.printStackTrace();
}
```

4. Saling berkirim dan menerima pesan

   Untuk mengirim pesan menggunakan DataOutput Stream yang telah dibentuk dan disambung pada buffer data output soket.

```
os.writeBytes(dataOutput);
```

   Untuk menerima pesan menggunakan BufferedReader yang telah dibentuk dan disambung dengan buffer data input soket.

```
dataInput=Is.readLine();
```

5. Menutup socket

   Pada client:

```
try {
  os.close();
  is.close();
  myKlien.close();
} catch (IOException io) {
  io.printStackTrace();
}
```

   Pada server:

```
try {
  os.close();
  is.close();
  layananSocket.close();
} catch (IOException ic) {
  ic.printStackTrace();
}
```

D. Membangun Aplikasi Client-Server TCP Sederhana

Demo D1-5

Buatlah aplikasi client server TCP sederhana. Server akan membuat soket server dan menerima permintaan koneksi dari satu client saja. Setelah itu server akan menunggu data yang dikirim oleh client. Jika pesan yang dikirim oleh client adalah “salam” maka server akan membalas mengirim pesan “salam juga” . Selain dari itu, server akan mengirim pesan “Maaf, saya tidak mengerti”.

1. Buat program server di bawah ini, simpan dengan nama **simpleServer.java**:

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

public class simpleServer {
  public final static int TESTPORT = 5000;
  public static void main(String args[]) {
    ServerSocket checkServer = null;
    String line;
    BufferedReader is = null;
    DataOutputStream os = null;
    Socket clientSocket = null;

    try {
      checkServer = new ServerSocket(TESTPORT);
      System.out.println("Aplikasi Server hidup ...");
    } catch (IOException e) {
      System.out.println(e);
    }

    try {
      clientSocket = checkServer.accept();
      is = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
      os = new DataOutputStream(clientSocket.getOutputStream());
    } catch (Exception ei) {
      ei.printStackTrace();
    }

    try {
      line = is.readLine();
      System.out.println("Terima : " + line);
      if (line.compareTo("salam") == 0) {
        os.writeBytes("salam juga");
      } else {
        os.writeBytes("Maaf, saya tidak mengerti");
      }
    } catch (IOException e) {
      System.out.println(e);
    }
```

```
    try {
      os.close();
      is.close();
      clientSocket.close();
    } catch (IOException ic) {
      ic.printStackTrace();
    }

  }
}
```

2. Buat program client di bawah ini, simpan dengan nama **simpleClient.java:**

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

public class simpleClient {
  public final static int REMOTE_PORT = 5000;
  public static void main(String args[]) throws Exception {
    Socket cl = null;
    BufferedReader is = null;
    DataOutputStream os = null;
    BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
    String userInput = null;
    String output = null;

    // Membuka koneksi ke server pada port REMOTE_PORT
    try {
      cl = new Socket(args[0], REMOTE_PORT);
      is = new BufferedReader(new
InputStreamReader(cl.getInputStream()));
      os = new DataOutputStream(cl.getOutputStream());
    } catch(UnknownHostException e1) {
      System.out.println("Unknown Host: " + e1);
    } catch (IOException e2) {
      System.out.println("Erorr io: " + e2);
    }

    // Menulis ke server
    try {
      System.out.print("Masukkan kata kunci: ");
      userInput = stdin.readLine();
      os.writeBytes(userInput + "\n");
    } catch (IOException ex) {
      System.out.println("Error writing to server..." + ex);
    }

    // Menerima tanggapan dari server
    try {
      output = is.readLine();
      System.out.println("Dari server: " + output);
    } catch (IOException e) {
      e.printStackTrace();
    }
```

```
    // close input stream, output stream dan koneksi
    try {
      is.close();
      os.close();
      cl.close();
    } catch (IOException x) {
      System.out.println("Error writing...." + x);
    }

  }
}
```

3. Kompilasi kedua program diatas dengan :

```
$ javac simpleServer.java
$ javac simpleClient.java
```

4. Jalankan kedua program tersebut dikomputer anda. Pertama jalankan server, (buka jendela console shell lebih dulu), dan tunggu koneksi client

```
$ java simpleServer
```

5. Untuk menjalankan program client buka jendela console shell baru dan ketikkan.

```
$ java simpleClient <nama-komputer-server>
```

6. Pada aplikasi client, masukkan kata kunci yang diminta, yaitu “salam”. Perhatikan apa terjadi kemudian di sisi aplikasi Server maupun client. Coba juga Anda memasukkan kata-kata yang lain.
7. Lakukan langkah 5 dan 6 dengan menjalankan aplikasi client dan server di komputer yang berbeda.

Latihan L1-1

Kembangkan aplikasi client server pada demo D1-1 sehingga aplikasi dan server dapat bertukar pesan terus menerus. Setiap kali aplikasi client mengirimkan suatu pesan, maka server akan membalikkan pesan tersebut dan mengirimkannya kembali ke client. Masing-masing aplikasi senantiasa menampilkan pesan yang diterimanya. Aplikasi client senantiasa meminta user memasukkan pesan yang hendak dikirim ke server. Kedua aplikasi selesai jika aplikasi client mengirim pesan “exit”. Beri nama file program **comServer.java** dan **comClient.java.**

Tampilan pada sisi client:

```
Aplikasi Client hidup ...
Masukkan pesan: salam
Balasan: malas
Masukkan pesan: hello
Balasan: olleh
Masukkan pesan: exit
Aplikasi Client selesai ...
```

Tampilan pada sisi server:

```
Aplikasi Server hidup ...
Pesan masuk: salam
Pesan masuk: hello
Pesan masuk: exit
Aplikasi Server selesai ...
```

E. Membangun Aplikasi Client untuk SMTP Server

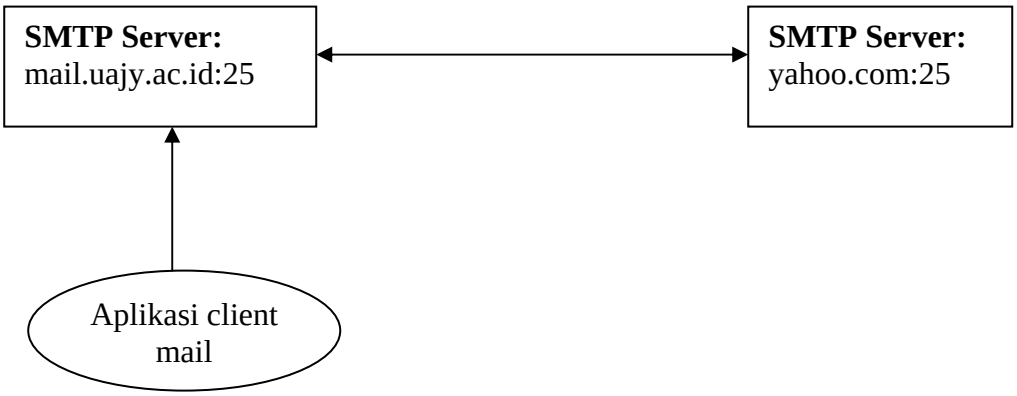
Dalam bagian ini, kita akan membuat aplikasi client untuk layanan server yang ada, yaitu layanan SMTP (Simple Mail Transfer Protocol) server. Apakah yang dibutuhkan agar dapat membangun suatu aplikasi client yang akan berkomunikasi dengan suatu layanan server yang sudah ada? Ada sejumlah hal yang harus diketahui oleh pembuat program client yaitu:

- Alamat soket layanan server tersebut (yaitu alamat IP dan nomer port layanannya)
- Protokol pesannya.

Setiap layanan server memiliki protokol pesan tersendiri. Sekalipun aplikasi client dapat mengirim pesan apa saja, namun tidak semua pesan itu dapat dimengerti oleh server. Hanya pesan-pesan yang dimengerti oleh aplikasi server saja yang akan diproses. Jadi aplikasi client haruslah mengetahui pesan-pesan yang dapat dimengerti oleh layanan server.

SMTP

SMTP adalah suatu protokol pesan untuk layanan pengiriman pesan email. Dewasa ini orang berkirim pesan email melalui aplikasi web ataupun menggunakan aplikasi mail client desktop (seperti outlook express, eudora light) untuk mengirim dan membaca email. Untuk dapat memahami apa yang sesungguhnya terjadi pada pengiriman email lihat gambar 4. Misalkan kita hendak hendak mengirim pesan ke budi@yahoo.com, maka aplikasi client kita minimal harus mengontak salah satu mail server yang ada diinternet. Misalnya lewat SMTP Server *mail.uajy.ac.id.* Mail server ini yang nantinya akan memforward pesan kita ke SMTP server yahoo.com yang mengelola account email budi@yahoo.com. Perhatikan untuk mengirim pesan ini, pengirim bahkan tidak membutuhkan account di SMTP Server *mail.uajy.ac.id.* Itu sebabnya hal ini memungkinkan terjadinya SPAM. Dengan kata lain, hal pertama yang dibutuhkan oleh aplikasi client adalah pengaksesan ke salah satu email server yang ada diinternet.



Gambar 4. Pengiriman email pada layanan SMTP

Hal kedua yang perlu diketahui oleh aplikasi client adalah format pertukaran pesan yang dapat dipahami oleh SMTP server. Untuk mengetahui hal ini, memang pemrogram harus membaca dokumentasi layanan suatu server terlebih dulu. Untuk mengerti pola

pesan yang dimengerti oleh SMTP server, kita dapat menggunakan tool sederhana yaitu telnet.

Telnet adalah program sederhana untuk membuka koneksi ke suatu soket. Setelah koneksi terbuka, maka pengguna dapat mengirimkan pesan-pesan ke soket tersebut. Demo berikut akan mencoba untuk mengirimkan pesan ke SMTP server dengan menggunakan tool telnet ini. Secara ringkas format pengiriman email ke SMTP server adalah:

HELO
MAIL from: <email pengirim>
RCPT to: <email tujuan>
DATA
From: <email pengirim yang akan tampil di pesan>
Subject: <judul pesan>
<isi pesan>
. (tanda titik untuk akhiri pesan)
QUIT (untuk mengakhiri koneksi)

Demo D1-6

1. Jalankan program telnet dan buka koneksi soket ke komputer *mail.uajy.ac.id* dan nomer port 25 (alamat komputer dapat diganti dengan nama SMTP server yang diketahui)

```
$ telnet mail.uajy.ac.id 25
```

2. Jalankan perintah-perintah dibawah ini dan amati pesan balasan dari SMTP server (pada contoh dibawah, semua perintah dari client diawali huruf Kapital, selebihnya adalah balasan dari SMTP server, jangan lupa beri pesan tanda titik setelah pesan bye. Ganti alamat RCPT TO: dengan alamat email anda sendiri)

```
220 mail.uajy.ac.id ESMTP
HELO
250 mail.uajy.ac.id
MAIL FROM:any@mail
250 ok
RCPT TO: kusnadi@mail.uajy.ac.id
250 ok
DATA
354 go ahead
FROM: any@mail
SUBJECT: testing
Hi, ini percobaan kirim email.
Bye.
.
250 ok 1184721672 qp 741
QUIT
```

```
221 mail.uajy.ac.id

Connection to host lost.
```

3. Coba apakah email sudah masuk ke account email tujuan menggunakan aplikasi client email berbasis web account tersebut.

Demo D1-7

Setelah mencoba mengirim pesan secara langsung lewat SMTP server menggunakan tools telnet. Saatnya untuk membuat aplikasi desktop sederhana untuk mengotomatisasi pengiriman pesan. Lakukan langkah berikut:

1. Buatlah program dibawah ini dan beri nama **smtpClient.java**

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

public class smtpClient{
    public static void main (String[] argv)
    {
            Socket smtpSocket=null;
            DataOutputStream os=null;
            BufferedReader is=null;

            try
            {
                    smtpSocket= new Socket("mail.uajy.ac.id",25);
                    os=new DataOutputStream(smtpSocket.getOutputStream());
                    is=new BufferedReader(new
                       InputStreamReader(smtpSocket.getInputStream()));
            }catch(UnknownHostException e){
                    System.err.println("Nama komputer tidak dikenali");
            }catch(IOException e){
                    System.err.println("Tidak dapat melakukan operasi io");
            }

            if(smtpSocket!=null && os!=null && is!=null)
            {
                    try{
                            //mengirim dan mencetak pesan
                            System.out.println(is.readLine());
                            os.writeBytes("HELO\r\n");
                                    System.out.print("HELO\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("MAIL From:any@mail\r\n");
                                    System.out.print("MAIL From:any@mail\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("RCPT To:kusnadi@mail.uajy.ac.id\r\n");
                                    System.out.print("RCPT
                                            To:kusnadi@mail.uajy.ac.id\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("DATA\r\n");
                                    System.out.print("DATA\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("From: any@mail\r\n");
                                    System.out.print("From: any@mail\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("Subject: testing\r\n");
                                    System.out.print("Subject: testing\n");
                                    System.out.println(is.readLine());
                            os.writeBytes("Hi, ini percobaan kirim email.\r\n");
                                    System.out.print("Hi, ini percobaan kirim
                                                    email.\n");
                                    System.out.println(is.readLine());
```

```
                        os.writeBytes("\r\n.\r\n");
                                System.out.print(".\n");
                                System.out.println(is.readLine());
                        os.writeBytes("QUIT\r\n");
                                System.out.print("QUIT\n");
                                System.out.println(is.readLine());

                        //menutup semua object
                        os.close();
                        is.close();
                        smtpSocket.close();


                } catch(UnknownHostException e){
                        System.err.println("Nama komputer tidak dikenali");
                }catch(IOException e){
                        System.err.println("Tidak dapat melakukan operasi io");
                }

        }

    }
}
```

2. Lakukan kompilasi dan jalankan program ditas..amati hasilnya..

Latihan L1-2

Kembangkan aplikasi D1-7 sehingga dapat menentukan, pada saat eksekusi, SMTP Server yang digunakan untuk merelay pesan email. Aplikasi juga meminta masukan email tujuan, email asal, subject, dan isi. Beri nama file programnya **MySmtpClient.java.** Contoh tampilan diberikan dibawah ini (huruf miring artinya masukan user)

```
Aplikasi client Email hidup ...
Nama SMTP Server relay: mail.uajy.ac.id
Email Pengirim: any@mail
Email Tujuan: kusnadi@mail.uajy.ac.id
Subject: testing
Isi:
Hi, ini percobaan kirim email.
Bye.
.
Aplikasi client Email selesai ...
```

F. Pengiriman Obyek lewat Soket

Sejauh ini, tutorial yang ada memberikan contoh mengirimkan pesan string lewat soket. Mungkinkah untuk mengirimkan obyek secara utuh lewat soket? Jawabanya adalah bisa. Mekanisme ini disebut dengan serialisasi. Syaratnya adalah kelas dari obyek yang hendak dikirimkan lewat soket haruslah mengimplementasikan *Serializable*. Perlu diperhatikan kelas yang mengimplementasikan Serializable tidak butuh melakukan *overriding* method apapun. Kelas interface Serializable terdapat pada paket Java.io.

Selain itu, obyek yang akan dikirim lewat soket haruslah menggunakan **ObjectOutputStream**, sedangkan untuk membacanya menggunak **ObjectInputStream.** Serialisasi ini bukan saja berguna untuk mengirim obyek lewat soket namun juga untuk menyimpan obyek secara persisten ke dalam file.

Demo D1-7

Tutorial ini akan membuat kelas Staff yang berisi informasi data pegawai. Data staff akan dikirimkan dari suatu aplikasi client ke aplikasi server lewat soket.

1. Buatlah kelas dibawah ini dan simpan sebagai **Staff.java**

```
import java.io.*;

public class Staff implements Serializable{
   String nama;
   String divisi;
   int    umur;

   public Staff(String nama, String divisi, int umur)
   {
         this.nama=nama;
         this.divisi=divisi;
         this.umur= umur;
   }

   public void print()
   {
         System.out.println("Data Staff: ");
         System.out.println("Nama : " + nama);
         System.out.println("Divis: " + divisi);
         System.out.println("Umur : "+ umur);
   }

}
```

2. Buatlah kelas dibawah ini dan simpanlah sebagai **ObjectClient.java:**

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

```
public class ObjectClient{
   private static int SRV_PORT = 5000;
   private static ObjectOutputStream os=null;

   public static void main(String argv[]) throws Exception{
         try{
               //membuat soket client
               Socket soketClient= new Socket("127.0.0.1", SRV_PORT);

               //membuat stream untuk pengiriman obyek
               os= new
                   ObjectOutputStream(soketClient.getOutputStream());

               //membuat obyek dan mengirimkannya lewat stream obyek
               Staff pegawai= new Staff("Hendry","IT",30);
               os.writeObject(pegawai);

               System.out.println("Client mengirim data pegawai:");
               pegawai.print();
         }
         catch(Exception e){
               e.printStackTrace();
         }
   }
}
```

3. Buatlah kelas server dibawah ini dan simpan sebagai **ObjectServer.java**

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

public class ObjectServer {
   private static int SRV_PORT=5000;
   private static ObjectInputStream is=null;

   public static void main(String argv[]) throws Exception{
         //membuat soket server dan menunggu koneksi
         ServerSocket soketServer= new ServerSocket(SRV_PORT);
         Socket soketClient= soketServer.accept();

         //membuat stream untuk baca obyek
         is= new ObjectInputStream(soketClient.getInputStream());

         //menunggu dan membaca obyek yang dikirimkan
         Staff pegawai= (Staff) is.readObject();

         System.out.println("Server menerima data Pegawai");
         pegawai.print();

   }
}
```

4. Kompilasi program server dan jalankan

```
$ javac ObjectServer.java
$ java ObjectServer
```

5. Kompilasi program client dan jalankan

```
$ javac ObjectClient.java
$ java ObjectClient
```

6. Perhatikan hasil eksekusi disisi client dan server (perhatikan ekseskusi server dibawah)

```
Server menerima data Pegawai
Data Staff:
Nama : Hendry
Divis: IT
Umur : 30
```

G. PROYEK

Proyek P1-1

Selidiki protokol POP3 yang merupakan protokol untuk mengambil pesan untuk suatu email account langsung dari Email Server (menggunakan port 110). Lalu buatlah program client Email untuk membaca email dari suatu account email. Beri nama file program anda dengan **MyPOP3Client.java**

Project P1-2

Gabungkan dan kembangkan program pada L1-2 (SMTP client) dan P1-1 (POP3 Client) sehingga aplikasi gabungan dapat digunakan untuk membaca email dari account dan juga untuk mengirim email ke account lainnya. Urutan interaksi penguna dan aplikasi adalah:

- user memasukkan nama server SMTP dan POP3
- user memasukkan account email dan passwordnya
- user memilih menu aplikasi
    - list semua pesan yang ada
    - membaca suatu pesan tertentu dari list
    - menulis dan mengirim email (lihat L1-2)
    - keluar dari aplikasi

Beri nama file program anda dengan **MyEmailClient.java**