NETWORK PROGRAMMING

Yuliana Setiowati
Politeknik Elektronika Negeri Surabaya

Protokol

- Protokol adalah suatu aturan atau mekanisme dimana dua komputer atau lebih dapat saling berinterkoneksi.
- Protokol mendefinisikan suatu format paket data yang akan dipertukarkan untuk menunjang mekanisme tersebut.
- Protokol yang banyak digunakan adalah TCP/IP.
- Internet menggunakan TCP/IP.

Pengantar Jaringan

- Fungsi utama protokol TCP/IP adalah menyediakan sebuah mekanisme komunikasi point to point.
- Komunikasi dibentuk dengan dua aliran data (stream). Satu stream membaca data (stream input) dari satu proses ke proses lain, sedangkan satunya lagi mengirim data (stream output).
- Artinya ada stream input dan stream output dari tiap proses untuk melakukan komunikasi.

Pengantar Jaringan

- Dalam jaringan komputer terdiri atas kumpulan komputer yang berdiri sendiri (otonom) dan saling bekerja sama
- 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 bukanlah hal yang mudah, cara lain terdapat layanan *Naming*, seperti DNS (Domain Naming Service)
- Tugas layanan naming semacam ini menerjemahkan alamat komputer yang *user-friendly,* seperti *www.google.com* menjadi alamat IP seperti 66.102.7.104.

Port

- Protokol TCP/IP memiliki *port* yang digunakan sebagai mekanisme untuk mengidentifikasi sebuah proses.
- Port merupakan pintu masuk datagram dan paket data.
- Port data dibuat mulai dari 0 sampai dengan 65.536.
- Alamat port 0 - 65535
  - Port 0 - 1023 → *well known port*
    Misal port 80 untuk HTTP
    21 untuk FTP
    110 untuk POP3
    23 untuk Telnet
    25 untuk SMTP
  - Port 1024 – 49151 → *registered port*
  - *Port* 49152 – 65535 → *Dynamic port / Private Port*

Tipe pemrograman jaringan

- Komunikasi Stream atau Connection Oriented communication
- Komunikasi datagram atau Connectionless communication

Connection Oriented

- Protokol standar : TCP (Transmission Control Protokol)
- 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 pengirim.
- Koneksi yang terbentuk pada TCP bersifat dua arah (*bidirectional*).

Connection Oriented

- **TCP** melakukan transmisi data per segmen, artinya paket data dipecah dalam jumlah yang sesuai dengan besaran paket, kemudian dikirim satu persatu hingga selesai.
- Agar pengiriman data sampai dengan baik, maka pada setiap paket pengiriman, TCP akan menyertakan nomor seri (sequence number).
- Pada sisi soket penerima, paket-paket data ini akan disimpan, diurutkan kembali, dan akhirnya digabungkan kembali menjadi data besar.
- TCP adalah protocol yang *reliable* yang senantiasa menunggu konfirmasi dari pihak soket penerima (soket penerima harus mengirim balik sebuah sinyal *ACKnowledge* dalam satu periode yang ditentukan).

Connection Oriented

- Bila pada waktunya sang penerima belum juga memberikan ACK, maka terjadi “ **time out** “ yang menandakan pegiriman paket gagal dan **harus diulang kembali**.
- Konsekuensinya adalah TCP menimbulkan overhead lalulintas jaringan yang tinggi
- Di Java diimplementasikan dengan class Socket dan ServerSocket.

Connectionless

- Standar protokol komunikasi datagram dikenal dengan UDP (*User Datagram Protocol)*
- Pada UDP, setiap kali paket data dikirim, informasi soket client turut dikirimkan. Mengirim paket secara individu.
- 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.
- Konsekuensinya adalah TCP menimbulkan overhead lalulintas jaringan lebih tinggi dibanding UDP.

Connectionless

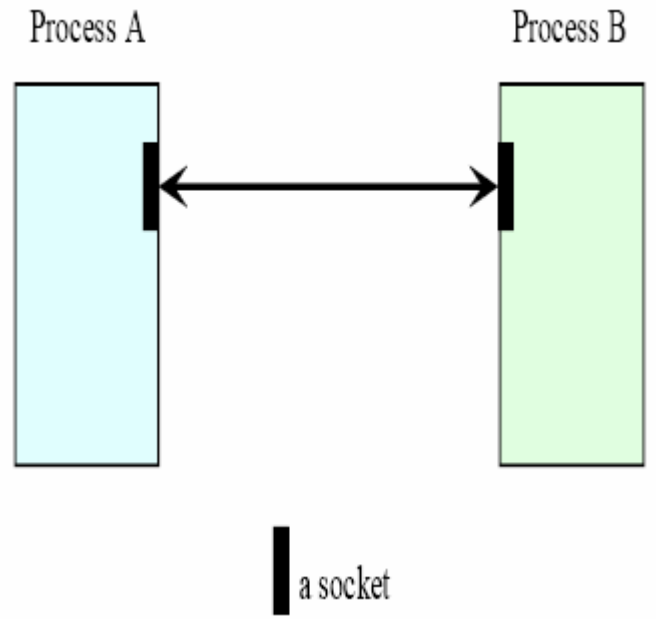
- Connectionless lebih cepat daripada Connection Oriented, namun connectionless tidak menjamin pengiriman.
- Di Java diimplementasikan dengan class DatagramPacket dan DatagramSocket.

Pemrograman Client/Server

- Suatu model umum yang diterapkan untuk pemrograman jaringan adalah model client/server.
- Melibatkan dua hal:
  - Client
  - Server
- Client : melakukan permintaan untuk suatu informasi atau mengirim sebuah perintah ke suatu aplikasi server.
- Server : menerima permintaan dari client, kemudian memproses berdasarkan permintaan tsb dan akan mengembalikan sesuatu ke client sebagai suatu hasil dari pemrosesan yang sudah dilakukan.

Socket

- Socket adalah sebuah abstraksi perangkat lunak yang digunakan sebagai suatu "terminal" dari suatu hubungan antara dua mesin atau proses yang saling berinterkoneksi. Di tiap mesin yang saling berinterkoneksi harus terpasang socket.



Socket

- Connection Oriented
  - Class yang mendukung adalah java.net.ServerSocket dan java.net.Socket
- Connectionless
  - Class yang mendukung adalah java.net.DatagramSocket

- Pada J2SE telah disediakan paket java.net yang berisi class dan interface yang menyediakan API (Application Programming Interface) level rendah (Socket, ServerSocket, DatagramSocket) dan level tinggi (URL, URLConnection).

Socket

- Agar suatu socket dapat berkomunikasi dengan socket lainnya, maka socket butuh diberi suatu alamat unik sebagai identifikasi.
- Alamat socket terdiri atas Alamat IP dan Nomer Port. Contoh alamat socket adalah **192.168.29.30**: **3000**, dimana nomer 3000 adalah nomer portnya. Alamat IP dapat menggunakan alamat Jaringan Lokal (LAN) maupun alamat 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.

Socket

- 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 socket haruslah mengikatkan dirinya dengan salah satu port pada komputernya. Pengikatan diri ini disebut dengan *binding*.

Class InetAddress

- Class yang berfungsi untuk mengambil informasi alamat IP suatu komputer.
- Class ini bersifat static dan tidak memiliki konstruktor.
- Menyediakan fungsi yang dapat digunakan untuk mendapatkan alamat IP.
  - getByName(namaHost)

    menerima sebuah string nama host dan mengembalikan alamat IP sesungguhnya berdasarkan DNS berupa object InetAddress. Untuk menampilkannya gunakan method toString().
  - getLocalHost()

    akan mengembalikan informasi alamat IP dan nama host pada komputer lokal.
  - getHostName()

    Gets the host name for this IP address
  - getHostAddress()

    Returns the IP address string in textual presentation.
  - getAllByName : akan mengembalikan informasi alamat IP dan nama host pada komputer lokal.

Class InetAddress

- Program di bawah ini untuk mengetahui nama komputer lokal.

```
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());
        }
}
```

```
Output - ProjectTeori (run-single)
init:
deps-jar:
compile-single:
run-single:
Nama komputer Anda :eepis-yuli
BUILD SUCCESSFUL (total time: 0 seconds)
```

Class InetAddress

- program untuk mendapatkan nama komputer dari Alamat IP.

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

public class IPtoName2{
   public static void main(String args[]){
      Scanner scan = new Scanner(System.in);
      System.out.println("Masukkan No IP : ");
      String host = scan.next() ;

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

```
run-single:
Masukkan No IP :
10.252.44.177
eepis-yuli.eepis-its.edu
```

Class InetAddress

- Buatlah program di bawah ini, masukkan misal www.detik.com maka akan ditampilkan Alamat IP dari www.detik.com. Masukkan host name : java.sun.com, berapakah Alamat IPnya?

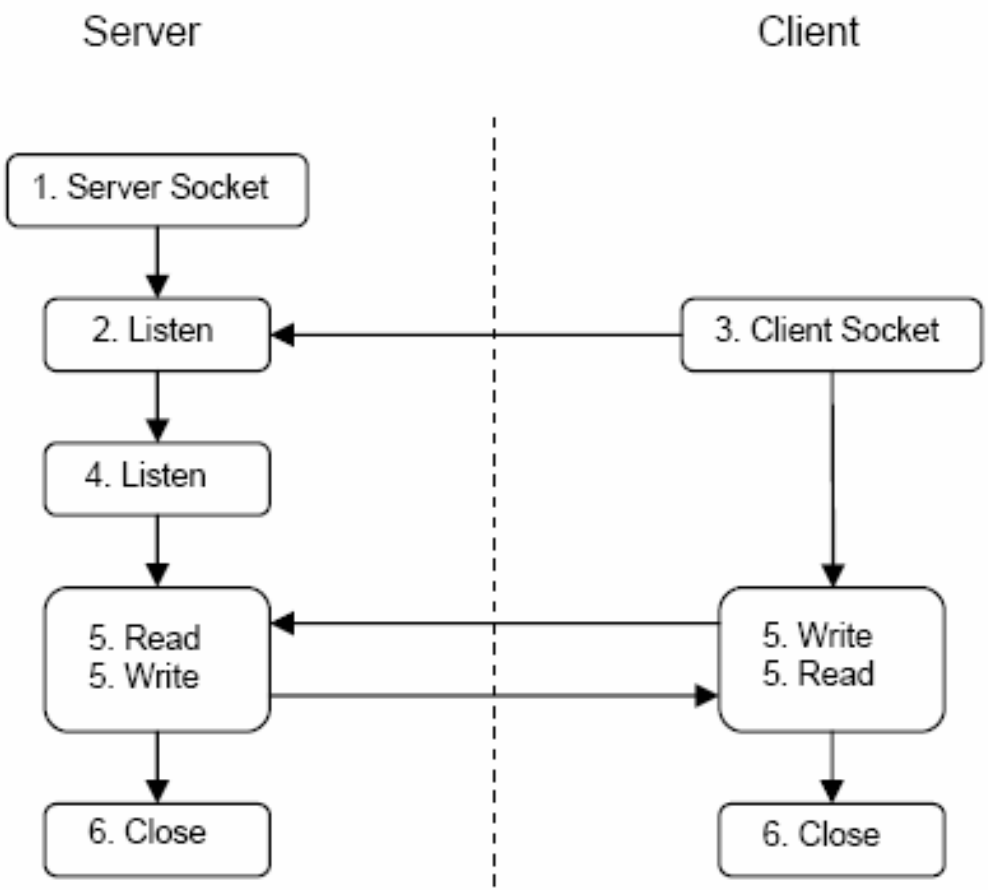
```
run-single:
Enter host name :
www.detik.com
IP Address : www.detik.com/203.190.241.43
```

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

public class IPFinder{
   public static void main(String args[]) throws IOException{
      String host;
      BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Enter host name : ");
      host = input.readLine();

      try{
         InetAddress address = InetAddress.getByName(host);
         System.out.println("IP Address : " + address);
      }catch(UnknownHostException e){
         System.out.println("Could not find " + host);
      }
   }
}
```

Model Aplikasi Client Server



Model Aplikasi Client Server

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

Pemrograman Socket TCP

- 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.

Komunikasi Socket TCP di Server

- Buatlah sebuah objek ServerSocket. Konstruktor ServerSocket memerlukan port number (1024-65535) sebagai argumen. Sebagai contoh :

  ```
  ServerSocket servSock = new ServerSocket(1234);
  ```

  Server akan menunggu koneksi dari client pada port 1234
- Server dalam kondisi menunggu (listen). Operasi ini pada intinya menunggu permintaan koneksi dari sisi client.

  ```
  Socket link = servSock.accept();
  ```

Komunikasi Socket TCP di Server

- Buat input dan output stream. Stream ini digunakan untuk berkomunikasi dengan client. Objek InputStreamReader digunakan untuk menerima respon dari client. Sedangkan PrintWriter untuk mengirimkan data ke client.

```
BufferedReader in = new BufferedReader(new
InputStreamReader(link.getInputStream()));
PrintWriter out = new
PrintWriter(link.getOutputStream(),true);
```

- Saling berkirim dan menerima pesan. Gunakan method readLine() untuk menerima data dan method println() untuk mengirim data.

```
out.println("Message " + numMessages + ":" + message);
message = in.readLine();
```

- Menutup socket

```
link.close();
```

Komunikasi Socket TCP di Client

- Bangun koneksi ke server. Buatlah sebuah objek Socket, yang mempunyai konstruktor dengan dua argumen:
  - IP address server
  - Port Number

  Port number untuk server dan client haruslah sama.

```
Socket link = new
Socket(InetAddress.getLocalHost(),1234);
```

- Buat input dan output stream. Stream ini digunakan untuk berkomunikasi dengan client. Objek InputStreamReader digunakan untuk menerima respon dari client. Sedangkan PrintWriter untuk mengirimkan data ke client.

```
BufferedReader in = new BufferedReader(new
InputStreamReader(link.getInputStream()));
PrintWriter out = new
PrintWriter(link.getOutputStream(),true);
```

Komunikasi Socket TCP di Client

- Saling berkirim dan menerima pesan. Gunakan method readLine() untuk menerima data dan method println() untuk mengirim data.

```
out.println("Message " + numMessages + ":" +
message);
message = in.readLine();
```

- Menutup socket

```
link.close();
```

Aplikasi TCP 1 – Aplikasi Server

- Buatlah program Server dan Client. Program server hanya bisa terkoneksi dengan 1 client.

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

public class TCPEchoServer{
   private static ServerSocket  servSock;
   private static final int PORT = 50000 ;
   public static void main(String args[]){
      System.out.println("Opening Port.....\n");
      try{
         servSock = new ServerSocket(PORT);
      }catch(IOException e){
         System.out.println("Unable to attach to port");
         System.exit(1);
      }

      do{
         run();
      }while(true);
   }
```

Aplikasi TCP 1 – Aplikasi Server

```
private static void run(){
   Socket link = null ;
   try{
       link = servSock.accept();
       BufferedReader in = new BufferedReader(new InputStreamReader(link.getInputStream()));
       PrintWriter out = new PrintWriter(link.getOutputStream(),true);

       int numMessages=0;
       String message=in.readLine();

       while(!message.equals("close")){
         System.out.println("Message received"+message);
         numMessages++;
         out.println("Message " + numMessages + ":" + message);
         message = in.readLine();
       }

       out.println(numMessages + " message received.");
   }catch(IOException e){
       e.printStackTrace();
   }
```

Aplikasi TCP 1 – Aplikasi Server

```
        finally{
            try{
                System.out.println("*********Closing Connection****");
                link.close();
            }
            catch(IOException e){
                System.out.println("Unable to disconnect");
                System.exit(1);
            }
        }
    }
}
```

Aplikasi TCP 1 – Aplikasi Client

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

public class TCPEchoClient{
    private static InetAddress host;
    private static final int PORT = 50000;
    public static void main(String args[]){
        try{
            host = InetAddress.getLocalHost();
        }catch(UnknownHostException e){
            System.out.println("Host ID Not Found");
            System.exit(1);
        }
        run();

    }
```

Aplikasi TCP 1 – Aplikasi Client

```
private static void run(){
   Socket link = null ;
   try{
     link = new Socket(host, PORT);
     BufferedReader in = new BufferedReader(new InputStreamReader(link.getInputStream()));
     PrintWriter out = new PrintWriter(link.getOutputStream(),true);
     BufferedReader userEntry = new BufferedReader(new InputStreamReader(System.in));
     String message, response ;

     do{
        System.out.print("Enter message : ");
        message = userEntry.readLine();
        out.println(message);
        response = in.readLine();
        System.out.println("SERVER " + response);
     }while(!message.equals("close"));
```

Aplikasi TCP 1 – Aplikasi Client

```
        }catch(IOException e){
          e.printStackTrace();
        }
        finally{
          try{
              System.out.println("closing connection");
              link.close();
          }
          catch(IOException e){
              System.out.println("Unable to disconnect!");
              System.exit(1);
          }
        }
    }
}
```

Menjalankan Aplikasi TCP 1

- Jalankan TCPEchoServer terlebih dahulu
- Selanjutnya jalankan TCPEchoClient tulislah message “Java”.
- Maka pada server akan menampilkan “Java”

SERVER

```
run-single:
Opening Port.....

Message received : Java
```

CLIENT

```
run-single:
Enter message : Java
SERVER Message 1:Java
Enter message :
```

Menjalankan Aplikasi TCP 1

- Tambahkan sebuah client lagi
- Apa yang terjadi ? Pada saat client 2 menuliskan message maka server tidak menerima message dari client 2

SERVER

```
run-single:
Opening Port.....

Message received : Java
```

CLIENT

```
                                ProjectTeori (r
init:
deps-jar:
compile-single:
run-single:
Enter message : client 2
```

Aplikasi TCP 2 (Multi Client) - Server

- Program server dapat menangani lebih dari 1 client

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

public class MultiEchoServer{
   private static ServerSocket  servSock;
   private static final int PORT = 1234 ;
   public static void main(String args[]) throws IOException{
      System.out.println("Opening Port.....\n");
      try{
         servSock = new ServerSocket(PORT);
      }catch(IOException e){
         System.out.println("Unable to attach to port");
         System.exit(1);
      }

      do{
         Socket client = servSock.accept();
         ClientHandler handler = new ClientHandler(client);
         handler.start();
      }while(true);
   }
}
```

Aplikasi TCP 1 (Multi Client) - Server

```
class ClientHandler extends Thread{
    private Socket client ;
    private BufferedReader in ;
    private PrintWriter out ;

    public ClientHandler(Socket socket){
        client = socket ;

        try{
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));
            out = new PrintWriter(client.getOutputStream(),true);
        }catch(IOException e){
            e.printStackTrace();
        }
    }
```

Aplikasi TCP 2 (Multi Client) - Server

```
public void run(){
   try{
      String received ;
      do{
         received = in.readLine();
         System.out.println(received);
         StringBuffer sb = new StringBuffer(received);
         out.println("ECHO : " + received);
      }while(!received.equals("QUIT"));
   }catch(IOException e){
      e.printStackTrace();
   }
   finally{
      try{
         if (client != null){
            System.out.println("Closing down connection");
            client.close();
         }
      }catch(IOException e){
         e.printStackTrace();
      }
   }
}
}
```

Aplikasi TCP 2 (Multi Client) -Client

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

public class MultiEchoClient{

   private static InetAddress host ;
   private static final int PORT = 1234;
   private static Socket link ;
   private static BufferedReader in ;
   private static PrintWriter out ;
   private static BufferedReader keyboard ;

   public static void main(String args[]){
      try{
         host = InetAddress.getLocalHost();
         //String strHost = args[0] ;
         //host = InetAddress.getByName(strHost);

         link = new Socket(host,PORT);

         in = new BufferedReader(new InputStreamReader(link.getInputStream()));
         out = new PrintWriter(link.getOutputStream(),true);

         keyboard = new BufferedReader(new InputStreamReader(System.in));

         String message, response;
```

```
            do{
                System.out.print("Enter message(QUIT to exit)");
                message = keyboard.readLine();
                out.println(message);
                response = in.readLine();
                System.out.println(response);
            }while(!message.equals("QUIT"));

        }catch(UnknownHostException e){
             System.out.println("Host ID not found!");
        }
        catch(IOException e){
            e.printStackTrace();
        }
        finally{
            try{
                if (link != null){
                    System.out.println("Closing down connection");
                    link.close();
                }
            }
            catch(IOException e){
                e.printStackTrace();
            }
        }
    }
}
```

Menjalankan Aplikasi TCP 2

- Jalankan MultiEchoServer terlebih dahulu
- Selanjutnya jalankan MultiEchoClient tulislah message “Client 1”.
- Maka pada server akan menampilkan “Client 1”
- Jalankan kembali MultiEchoClient tulislah message “Client 2”.
- Maka pada server akan menampilkan “Client 2”

SERVER

```
run-single:
Opening Port.....

CLIENT 1
CLIENT 2
```

CLIENT 1

```
run-single:
Enter message(QUIT to exit)CLIENT 1
ECHO : CLIENT 1
Enter message(QUIT to exit)
```

CLIENT 2

```
run-single:
Enter message(QUIT to exit)CLIENT 2
ECHO : CLIENT 2
Enter message(QUIT to exit)
```

Komunikasi Socket Datagram

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

Komunikasi Socket Datagram di Server

- Buatlah sebuah objek DatagramSocket. Konstruktor DatagramSocket mempunyai satu argument yaitu no port.

```
DatagramSocket dgramSocket = new DatagramSocket(1234);
```

- Buat buffer untuk datagram yang masuk.

```
byte[] buffer = new byte[256];
```

- Buatlah sebuah objek DatagramPacket untuk datagram yang masuk. Konstruktor dari objek ini memerlukan dua argumen yaitu array byte dan besar dari array ini

```
DatagramPacket inPacket = new
DatagramPacket(buffer,buffer.length);
```

- Menerima datagram yang masuk

```
dgramSocket.receive(inPacket);
```

- **Menerima alamat pengirim dan port yang masuk (dari client)**

```
InetAddress clientAddress = inPacket.getAddress();
int clientPort = inPacket.getPort();
```

Komunikasi Socket Datagram di Server

- Mengambil data dari buffer. Untuk memudahkan penanganan, maka data di ambil sebagai string. Konstruktor String yang digunakan mempunyai 3 argumen yaitu
  - Array dengan tipe byte
  - Posisi awal array yang akan diambil(posisi 0)
  - Jumlah byte yang akan diambil (sama dengan besar buffer)

```
String   messageIn = new
String(inPacket.getData(),0,inPacket.getLength());
```

- Buatlah datagram respon. Buatlah sebuah objek DatagramPacket, dengan konstruktor yang memerlukan 3 argument.
  - Array dengan tipe byte yang berisi message response
  - Besar response
  - Alamat client
  - No port

```
DatagramPacket outPacket = new
DatagramPacket(messageOut.getBytes(), messageOut.length(),
clientAddress, clientPort);
```

- Kirim datagram response

```
dgramSocket.send(outPacket);
```

- Close DatagramSocket.

```
dgramSocket.close();
```

Komunikasi Socket Datagram di Client

1. Buatlah sebuah objek DatagramSocket. Pembuatan objek DatagramSocket hampir sama seperti pada program server tapi untuk client tidak menggunakan no port

```
DatagramSocket dgramSocket = new DatagramSocket();
```

2. Buatlah datagram yang akan keluar (disertakan alamat client). Lihat langkah 7 pada server.

```
DatagramPacket outPacket = new
DatagramPacket(messageOut.getBytes(),
messageOut.length(), clientAddress, clientPort);
```

3. Kirim message datagram.

```
dgramSocket.send(outPacket);
```

4. Buatlah sebuah buffer untuk datagram yang masuk.

```
byte[] buffer = new byte[256];
```

5. Buatlah sebuah objek DatagramPacket untuk datagram yang masuk. Konstruktor dari objek ini memerlukan dua argumen yaitu array byte dan besar dari array ini

```
DatagramPacket inPacket = new
DatagramPacket(buffer,buffer.length);
```

Komunikasi Socket Datagram di Client

6. Menerima datagram yang masuk

```
dgramSocket.receive(inPacket);
```

7. Mengambil data dari buffer. Untuk memudahkan penanganan, maka data di ambil sebagai string. Konstruktor String yang digunakan mempunyai 3 argumen yaitu
   - Array dengan tipe byte
   - Posisi awal array yang akan diambil(posisi 0)
   - Jumlah byte yang akan diambil (sama dengan besar buffer)

```
String   messageIn = new
String(inPacket.getData(),0,inPacket.getLength());
```

8. Close DatagramSocket.

```
dgramSocket.close();
```

Aplikasi UDP - UDPEchoServer

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

public class UDPEchoServer{
    private static final int PORT = 1234 ;
    private static DatagramSocket dgramSocket ;
    private static DatagramPacket inPacket, outPacket ;
    private static byte[] buffer  ;

    public static void main(String args[]){
        System.out.println("Opening Port.....\n");
        try{
            dgramSocket = new DatagramSocket(PORT);
        }catch(SocketException e){
            System.out.println("Unable to attachto port !");
            System.exit(1);
        }
        run();
    }
```

Aplikasi UDP - UDPEchoServer

```
private static void run(){
    try{
        String messageIn, messageOut;
        int numMessages=0;
        do{
           buffer = new byte[256] ;
           inPacket = new DatagramPacket(buffer,buffer.length);
           dgramSocket.receive(inPacket);
           InetAddress clientAddress = inPacket.getAddress();
           int clientPort = inPacket.getPort();

           messageIn = new String(inPacket.getData(),0,inPacket.getLength());
           System.out.println("Message Received");

           numMessages++;
           messageOut = ("Message " + numMessages + ":" + messageIn);
           outPacket = new DatagramPacket(messageOut.getBytes(), messageOut.length(), clientAddress, clientPort);
           dgramSocket.send(outPacket);
        }while(true);
    }
```

Aplikasi UDP - UDPEchoServer

```
        catch(IOException e){
            e.printStackTrace();
        }
        finally
        {
            System.out.println("\nClosing Connection...");
            dgramSocket.close();
        }
}
```

Aplikasi UDP - UDPEchoClient

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

public class UDPEchoClient{
    private static InetAddress host ;
    private static final int PORT = 1234 ;
    private static DatagramSocket dgramSocket ;
    private static DatagramPacket inPacket, outPacket ;
    private static byte[] buffer  ;

    public static void main(String args[]){
        try{
            host = InetAddress.getLocalHost();
        }
        catch(UnknownHostException e){
            System.out.println("Host ID Not Found");
            System.exit(1);
        }
        run();
    }
```

```
private static void run(){
    try{
        dgramSocket = new DatagramSocket();
        BufferedReader userEntry = new BufferedReader(new InputStreamReader(System.in));
        String message="", response="";
        do{
            System.out.println("Enter message:");
            message = userEntry.readLine();
            if (!message.equals("CLOSE")){
                outPacket = new DatagramPacket(message.getBytes(),message.length(),host,PORT);
                dgramSocket.send(outPacket);
                buffer = new byte[256];
                inPacket = new DatagramPacket(buffer, buffer.length);
                dgramSocket.receive(inPacket);
                response = new String(inPacket.getData(),0,inPacket.getLength());
                System.out.println("SERVER :" + response);
            }
        } while(!message.equals("CLOSE"));
    }
    catch(IOException e){
         e.printStackTrace();
    }finally{
        System.out.println("Closing Connection");
        dgramSocket.close();
    }
}
}
```

Menjalankan Aplikasi UDP

- Jalankan UDPEchoServer terlebih dahulu
- Selanjutnya jalankan UDPEchoClient tulislah message “Client 1”.
- Maka pada server akan menampilkan “Message Received”

SERVER

```
run-single:
Opening Port.....

Message Received
```

CLIENT

```
run-single:
Enter message:
CLIENT 1
SERVER :Message 1:CLIENT 1
Enter message:
```