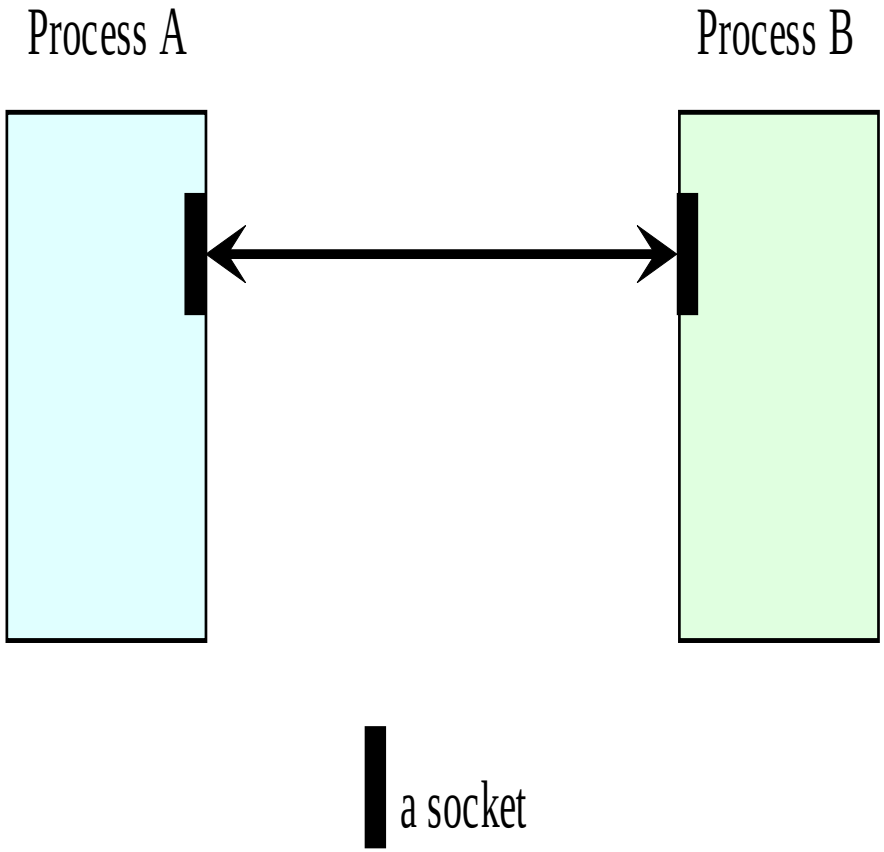
Sistem Terdistribusi 4

Pemrograman Socket

Socket

- Socket adalah sebuah abstraksi perangkat lunak yang digunakan sebagai suatu "terminal" dari suatu hubungan antara dua mesin atau proses yang saling berinterkoneksi.
- **End to end communication**



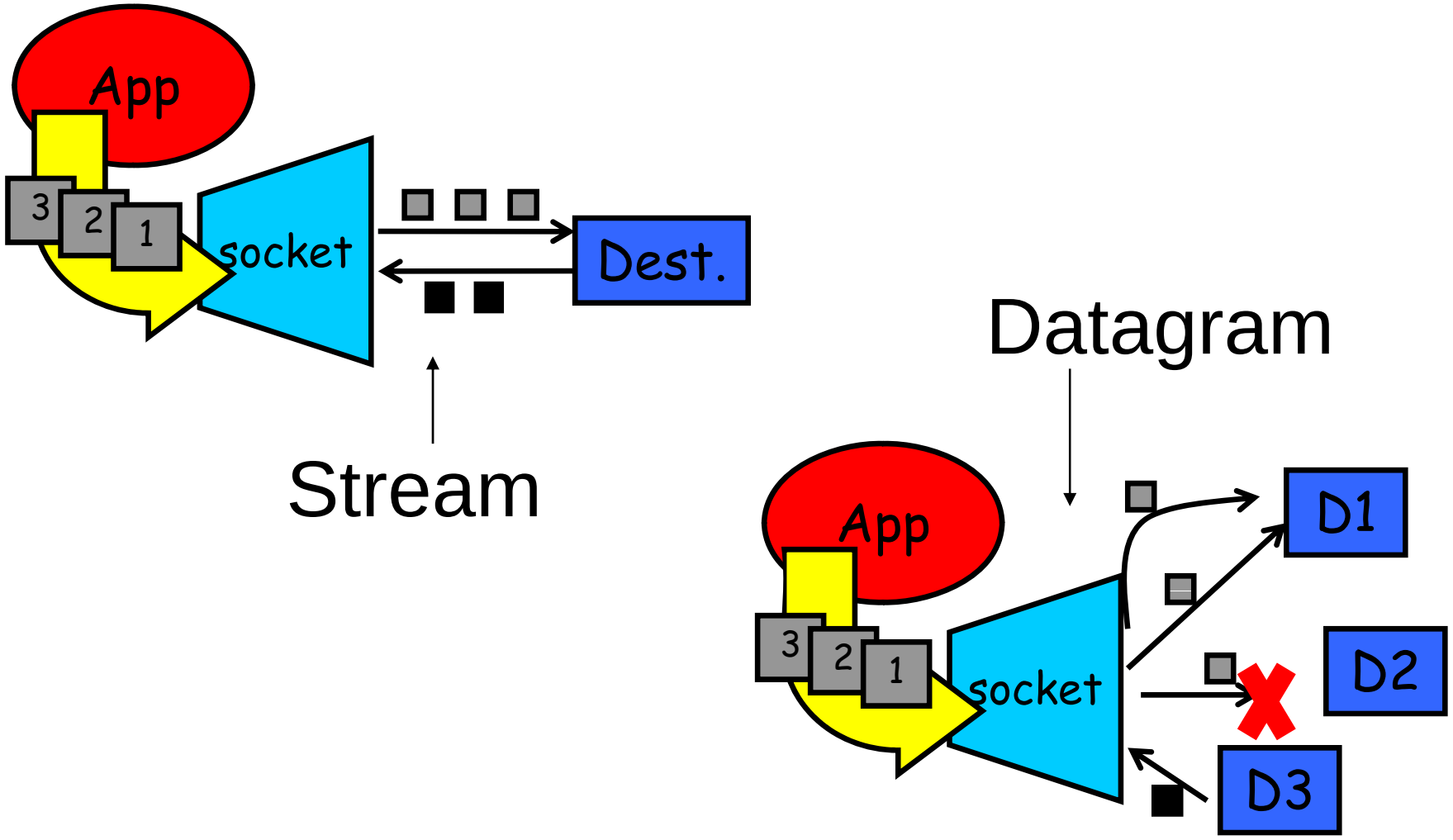
Addresses, Ports and Sockets

- Like apartments and mailboxes
  - Apartement is the **application**
  - Your apartment building address is the **address**
  - Your mailbox is the **port**
  - The post-office system is the **network**
  - The socket is the key that gives you access to the **right** mailbox
- Q: How do you choose which **port** a socket connects to?
  - Sesuai kesepakatan

Socket types

- **Stream socket**
  - Paket-paket data **byte**, bidirectional connection
  - **Ordered**, reliable delivering of packets
  - Use TCP
- **Datagram socket**
  - There is no connection.
    - Each packet is sent **independently** from the others
  - **Unordered**, unreliable delivering of packets
  - Use UDP
- **Raw sockets**
  - Out of our scope

Stream vs Datagram



Operasi Socket

- Socket dapat melakukan operasi:
  - Koneksi ke mesin remote (connect)
  - Mengirim data (send)
  - Menerima data (receive)
  - Mentutup koneksi (close)
  - Bind to a port (bind), khusus server
- Di tiap mesin yang saling berinterkoneksi, harus terpasang socket.
  - Dalam arti saling tahu sama tahu
  - Sesuai kesepakatan

Stream Socket lifetime

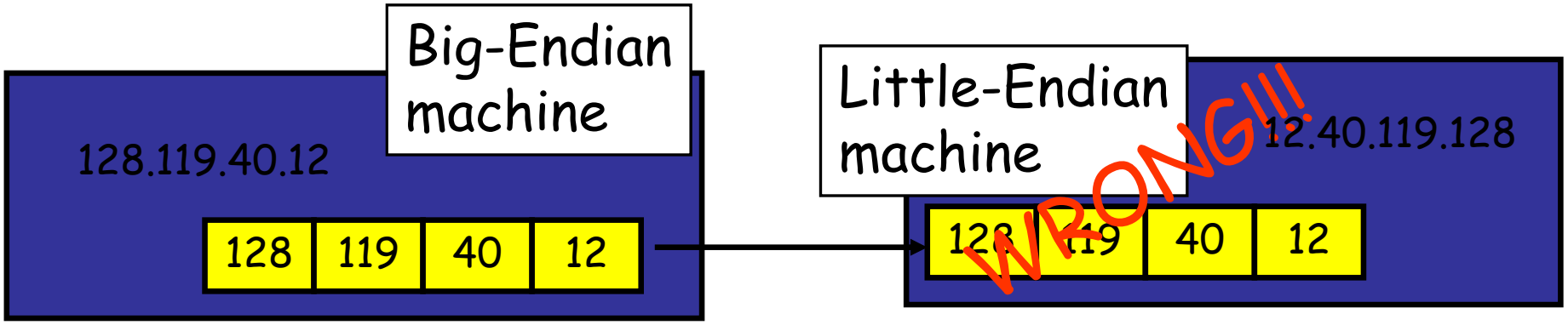
- Creation
- Binding
  - **Assigning a name** to the socket – until a name is assigned, no messages may be received.
  - Communicating processes are bound by an *association*, which in Internet is composed of **local and foreign addresses, and local and foreign ports**.
  - The ***bind()*** system call specifies half of an association {local address, local port},
  - while the ***connect*** and ***accept*** primitives complete the association {foreign address, foreign port}.

Stream Socket lifetime (cont.)

- Connection
  - Connection establishment, between a **server** and a **client**
  - The server ***binds*** a socket to a well known address and passively ***listens***, which means that he waits for a client to ***connect***
  - When a client connects, the server ***accepts*** the connection.
- Data transfer
  - When the two process are connected, **data flow** may begin between them (in byte data)
- Discard
  - When the communication ends, the sockets must be ***closed***, to enable **the system to release resources**, especially the bounded names (e.g. local ports) because they cannot be reused until they are available from any possible association.

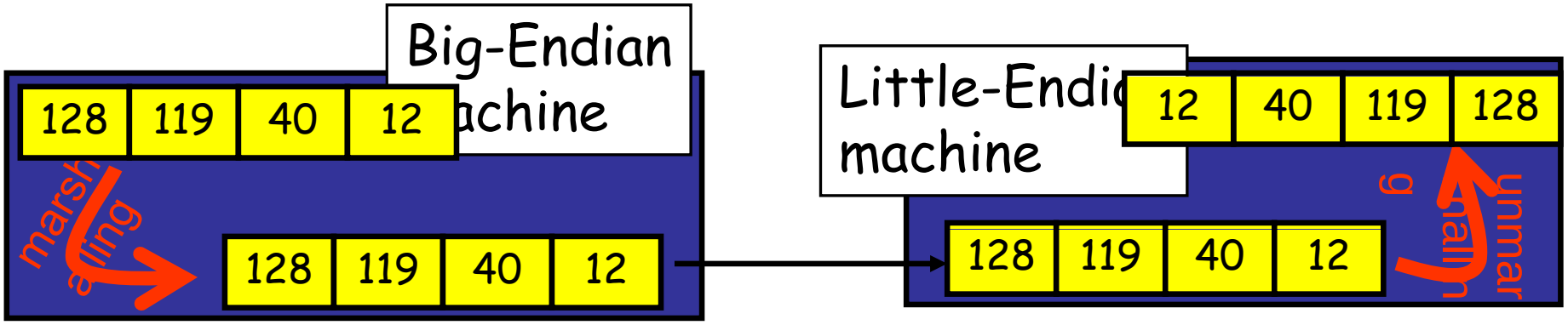
Data transfer

- Heterogenity
- Big Endian vs Little Endian
- Solution:
  - Marshalling
  - Unmarshalling



Marshalling / Unmarshalling

- Object Serialization



Serializable

- Pada pemrograman socket biasanya yang dikirim adalah **data stream**.
  - Nah, bagaimana jika yang dikirim adalah suatu **obyek**?
- Hal ini dapat dilakukan dengan menggunakan serialisasi obyek.
- *Object Serialization* adalah teknik dimana suatu program dapat menyimpan status obyek ke dalam sebuah **file** dan kemudian dapat dipanggil kembali dari file ke memori atau dikirim melalui jaringan.
  - Serialisasi memastikan agar obyek-obyek dapat sinkron
- Jika sebuah obyek ingin diserialisasi, maka obyek itu harus mengimplementasikan **java.io.Serializable**.
- Untuk menuliskan obyek yang terserialisasi ke file dibutuhkan I/O stream khusus, yaitu menggunakan *ObjectOutputStream* yang merupakan subclass dari *FilterOutputStream*.

Contoh Pegawai

```
import java.io.*;

public class Pegawai implements Serializable {
	private String nama;
	private int umur;
	private int gaji;

	public Pegawai(String nama, int umur, int gaji) {
		this.nama = nama;
		this.umur = umur;
		this.gaji = gaji;
	}

	public void print() {
		System.out.println("Data untuk " + this.nama);
		System.out.println("nama " + this.nama);
		System.out.println("umur " + this.umur);
		System.out.println("gaji " + this.gaji);
	}
}
```

Contoh SimpanPegawai

```
import java.io.*;

public class SimpanPegawai {
  public static void main(String[] args) {
    Pegawai aaa = new Pegawai("aaa", 28, 100);
    Pegawai bbb = new Pegawai("bbb", 30, 150);

    try {
      FileOutputStream fos = new FileOutputStream("db");
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(aaa);
      oos.writeObject(bbb);
      oos.flush();
    }
    catch (IOException e) {
      System.out.print("error " + e);
      System.exit(1);
    }
  }
}
```

Penjelasan

- Berarti program SimpanPegawai akan menyimpan 2 obyek pegawai yaitu “aaa” dan “bbb” ke dalam file bernama “db”.
- Sedangkan untuk pembacaan file yang berisi obyek juga harus dilakukan secara berurutan, yaitu “aaa” dulu baru “bbb”.
- Karena pembacaan dengan menggunakan **readObject()** yang mengembalikan Object, maka harus dilakukan casting sesuai dengan tipe Objectnya.

Contoh: BacaPegawai

```
import java.io.*;

public class BacaPegawai {
   public static void main(String[] args) {
      try {
         FileInputStream fis = new FileInputStream("db");
         ObjectInputStream ois = new ObjectInputStream(fis);

         Pegawai aaa = (Pegawai) ois.readObject();
         Pegawai bbb = (Pegawai) ois.readObject();

         aaa.print();
         bbb.print();
      }
      catch (IOException e) {
         System.exit(1);
      }
      catch (Exception e) {
         System.exit(1);
      }
   }
}
```

```
Data untuk aaa
nama aaa
umur 28
gaji 100
Data untuk bbb
nama bbb
umur 30
gaji 150
```

Serialisasi dgn Vector

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

public class VectorSerial {

    public static void main(String args[]) throws IOException,ClassNotFoundException {

      ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("filepeg.dat"));
      Vector<Pegawai> staffout = new Vector<Pegawai>();
      Vector<Pegawai> staffin = new Vector<Pegawai>();

      Pegawai[] staff = {new Pegawai("anton",20,1000), new Pegawai("yuan",21,1500), new Pegawai("Mahas",22,2000) };

      for (int i = 0; i<staff.length; i++)
        staffout.add(staff[i]);
      oos.writeObject(staffout);
      oos.close();

      ObjectInputStream ois = new ObjectInputStream(new FileInputStream("filepeg.dat"));
      try {
        staffin = (Vector<Pegawai>)ois.readObject();
        for(Pegawai p:staffin){
          p.print();
        }
    }
    catch (EOFException ex) {
      System.out.println (ex.getMessage());
      ois.close();
    }

    System.console().readLine();
    }
}
```

Transient

- Dalam contoh-contoh diatas, semua atribut dari kelas Pegawai secara otomatis bisa dibaca karena bersifat **serializable**.
- Kita bisa membatasi hak akses terhadap atribut tertentu saja yang bersifat serializable.
- Hal ini dapat dilakukan dengan menggunakan keyword ***transient***.
- Dengan keyword itu maka atribut tersebut tidak ikut “terbawa” untuk dikirimkan dalam deretan byte melalui I/O Stream.

Pegawai yang Transient

```
Data untuk aaa
nama aaa
umur 0
gaji 100
Data untuk bbb
nama bbb
umur 0
gaji 150
```

```
import java.io.*;

public class Pegawai implements Serializable {
	private String nama;
	private transient int umur;
	private int gaji;

	public Pegawai(String nama, int umur, int gaji) {
		this.nama = nama;
		this.umur = umur;
		this.gaji = gaji;
	}

	public void print() {
		System.out.println("Data untuk " + this.nama);
		System.out.println("nama " + this.nama);
		System.out.println("umur " + this.umur);
		System.out.println("gaji " + this.gaji);
	}
}
```

Stream sockets: Java Example

- package **java.net**
- Classes:
  - InetAddress
  - Socket
  - ServerSocket
  - DatagramSocket

InetAddress class

- Kelas ini digunakan untuk mengambil informasi IP suatu komputer.
  - Kelas ini bersifat **static** dan tidak memiliki konstruktor.
- Method-methodnya adalah:
  - **getByName(namahost)** yang akan menerima sebuah string nama host dan mengembalikan alamat IP berdasarkan DNS, berupa object InetAddress.
    - Untuk menampilkannya: gunakan method toString()
  - **getLocalHost()** yang akan mengembalikan alamat IP dan nama host pada komputer lokal.
  - **getAllByName(namahost)** mengembalikan array InetAddress
- Kemungkinan error: **UnknownHostException**

Contoh getByName

```
Enter host name: www.google.com
IP address: www.google.com/66.249.89.99
Press any key to continue...
```

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

public class IPFinder
{
    public static void main(String[] args)
    {
        String host;
        Scanner input = new Scanner(System.in);

        System.out.print("\n\nEnter host name: ");
        host = input.next();
        try
        {
            InetAddress address =
                              InetAddress.getByName(host);
            System.out.println("IP address: "
                                    + address.toString());
        }
        catch (UnknownHostException uhEx)
        {
            System.out.println("Could not find " + host);
        }
    }
}
```

Class **ServerSocket**

- Constructor
  - **ServerSocket**(int port)
  - **ServerSocket**(int port, int backlog)
  - **ServerSocket**(int port, int backlog, InetAddress bindAddr)
- Methods
  - Socket **accept**()
  - void **close**()
  - InetAddress **getInetAddress**()
  - int **getLocalPort**()

Class **Socket**

- Constructor
  - **Socket** (InetAddress address, int port)
  - **Socket** (String host, int port)
- Methods
  - InputStream **getInputStream**()
  - OutputStream **getOutputStream**()
  - void **close**()
  - InetAddress **getInetAddress**()
  - int **getLocalPort**()

Prinsip ServerSocket

- Create a **ServerSocket** object.
  - ServerSocket servSock = new ServerSocket(1234);
- Put the server into a **waiting** state.
  - Socket link = servSock.accept();
- Set up **input** and **output** streams.
  - Scanner input = new Scanner(link.getInputStream());
  - PrintWriter output = new PrintWriter(link.getOutputStream(),true);
- **Send** and **receive** data.

  output.println("Awaiting data...");

  String input = input.nextLine();
- **Close** the connection (after completion of the dialogue).
  - link.close();

Prinsip Socket (client)

- Establish a **connection** to the server.
  - the server's IP address (of type InetAddress);
  - the appropriate port number for the service.

    Socket link = new Socket(InetAddress.getLocalHost(),1234);
- Set up **input** and **output** streams.
  - Scanner input = new Scanner(link.getInputStream());
  - PrintWriter output = new PrintWriter(link.getOutputStream(),true);
- **Send** and **receive** data.
  - The Scanner object at the client will receive messages sent by the PrintWriter object at the server,
  - while the PrintWriter object at the client will send messages that are received by the Scanner object at the server (using methods nextLine and println respectively).
- **Close** the connection.

Contoh InfoClient dan InfoServer

- InfoClient.java
- InfoServer.java

InfoClient.java

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

public class InfoClient {
   private final int INFO_PORT=50000;
   private final String TargetHost = "localhost";
   private final String QUIT = "QUIT";

   /** Creates a new instance of InfoClient */
   public InfoClient() {
      try {
         /* siapkan input stream dari
            standard input (keyboard) */
         BufferedReader inFromUser =
            new BufferedReader(new
               InputStreamReader(System.in));

         /* buat koneksi socket ke server dan
            port yang dituju */
         Socket clientSocket = new
               Socket(TargetHost, INFO_PORT);
         /* siapkan output stream ke socket */
         DataOutputStream outToServer =
            new DataOutputStream(
               clientSocket.getOutputStream());

         /* siapkan input stream dari socket */
         BufferedReader inFromServer =
            new BufferedReader(
               new InputStreamReader(
                  clientSocket.getInputStream()));
```

```
/* tampilkan informasi welcome dari server
 * oleh karena method readLine() akan
 * membaca satu baris string
 * yang diakhir dengan karakter ENTER,
 * maka perlu diberikan 3
 * pemanggilan method readLine()
 * untuk membaca 3 baris pesan welcome
 * yang dikirim dari server
 */
System.out.println(inFromServer.readLine());
System.out.println(inFromServer.readLine());
System.out.println(inFromServer.readLine());
System.out.println("");

boolean isQuit = false;
while (!isQuit) {
   /* menunggu masukan perintah dari user */
   System.out.print("Perintah Anda : ");
   String cmd = inFromUser.readLine();

   /* konversi ke Upper dan cek
    * apakah perintah QUIT yang diketikkan
    * jika QUIT, beri nilai true untuk isQUIT
    * agar looping !isQUIT selesai
    */
   cmd = cmd.toUpperCase();
   if (cmd.equals(QUIT)) {
      isQuit = true;
   }

   /* kirim perintah yang dimasukkan ke server
    * dan diakhiri dengan karakter ENTER, karena
    * pada server data dari client dibaca
    * dengan method readLine() untuk membaca
    * satu baris string sampai dengan ENTER
    */
   outToServer.writeBytes(cmd + "\n");
   /* block reading ....
    * client harus menunggu balasan dari server
    */
   String result = inFromServer.readLine();
   System.out.println("Dari Server: " + result);
```

```
        }

        /* tutup semua stream dan koneksi socket */
        outToServer.close();
        inFromServer.close();
        clientSocket.close();
      }
      catch (IOException ioe) {
         System.out.println("Error:" + ioe);
      }
      catch (Exception e) {
         System.out.println("Error:" + e);
      }
   }

   /**
    * Program Utama InfoClient
    */
   public static void main(String[] args) {
     new InfoClient();
   }
}
```

InfoServer.java

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

public class Infoserver {
   private final int INFO_PORT=50000;
   private String datafromClient;

   /** InfoServer Constructor */
   public Infoserver() {
      BufferedReader inFromClient;
      DataOutputStream outToClient;
      Socket serverSocket;

      try {
         /* bind port 50000 ke alamat lokal */
         ServerSocket infoServer =
               new ServerSocket(INFO_PORT);

         System.out.println("Server telah siap...");

         /* lakukan perulangan tanpa henti,
          * sampai client memberikan perintah QUIT
          */
         while (true) {
            /* masuk ke mode listening,
             * server siap menerima permintaan dari client
             */
            serverSocket = infoServer.accept();
            System.out.println("Ada client " +
                               "yang terkoneksi!");

            /* buat input stream dari socket
             * dan juga sekaligus konversi dari
             * byte stream ke character stream
             * (InputStreamReader)
             * BufferedReader akan memudahkan
             * dalam pengolahan data karakter
             */
            inFromClient =
                  new BufferedReader(
                  new InputStreamReader(
                        serverSocket.getInputStream()));
```

```
45            /* buat output stream ke socket */
46            outToClient =
47                    new DataOutputStream(
48                        serverSocket.getOutputStream());
49
50            /* tulis welcome ke client */
51           outToClient.writeBytes("InfoServer versi 0.1\n"+
52                                  "hanya untuk testing..\n"+
53         "Silahkan berikan perintah TIME | NET | QUIT\n");
54
55           /* lakukan perulangan sampai client
56            * mengirimkan perintah QUIT
57            */
58           boolean isQUIT = false;
59           while (!isQUIT) {
60             /* baca data dari client */
61             datafromClient = inFromClient.readLine();
62
63             if (datafromClient.startsWith("TIME")) {
64               outToClient.writeBytes(new
65                       Date().toString() + "\n");
66             } else if (datafromClient.startsWith("NET")) {
67               outToClient.writeBytes(
68         InetAddress.getByName("localhost").toString() +
69                   "\n");
70             } else if (datafromClient.startsWith("QUIT"))
71             {
72                 isQUIT = true;
73             }
74           }
75           outToClient.close();
76           inFromClient.close();
77           serverSocket.close();
78
79           System.out.println("Koneksi client tertutup..");
80         }
81       }
82       catch (IOException ioe) {
83         System.out.print("error: " + ioe);
84       }
85       catch (Exception e) {
86         System.out.print("error: " + e);
87       }
```

```
    }

    /* program utama */
    public static void main(String[] args) {
        new InfoServer();
    }
}
```

Kita Lanjutkan dengan Socket UDP!

Bagaimana jika client yang dilayani lebih dari satu?

- **MULTI THREADING**

Kelas java.net.DatagramSocket

- Kelas ini mengirim dan menerima **DatagramPacket** dari atau ke jaringan.

Constructor:

- **DatagramSocket**(int port)
  - Kelas ini dapat digunakan untuk menyatakan penggunaan suatu nomor port sebagai "pintu" untuk menerima koneksi dari client.
- **DatagramSocket**(int port, InetAddress addr)
  - Kelas ini membentuk koneksi dengan protokol UDP pada alamat IP lokal tertentu dan pada nomor port tertentu.
- **DatagramSocket**()
  - Kelas ini membentuk koneksi dengan protokol UDP pada alamat IP lokal host dengan penentuan nomor portnya secara random berdasar tersedianya nomor port yang dapat digunakan.

Methods DatagramSocket

- **send**(DatagramPacket data), akan mengirim DatagramPacket ke host dan port yang dituju
- **receive**(DatagramPacket data), akan memblok eksekusi sampai suatu paket lengkap diterima

Kelas java.net.DatagramPacket

- Merupakan kelas yang menyatakan atau mewakili sebuah **paket informasi**, yaitu sebuah **array byte** yang dipertukarkan pada jaringan.

Constructor:

- **DatagramPacket**(byte[] buf, int length)
  - Kelas ini dapat digunakan untuk mengambil informasi. Constructor ini membutuhkan sebuah array byte yang menjadi parameter pertama, yang berfungsi untuk menyimpan data dan informasi ukuran data yang diterima.
- **DatagramPacket**(byte[] buf, int length, InetAddress address, int port)
  - Constructor ini digunakan untuk membuat paket Datagram yang akan mengirim data. Constructor ini memerlukan informasi array byte yang akan dikirim dan panjangnya, serta alamat dan port yang dituju.

Methods DatagramPacket

- **getData**(), untuk mengambil informasi data, kembaliannya berupa byte[] akan mempersulit kita membaca data, sehingga kita bisa mengubahnya menjadi String atau ByteArrayInputStream
- **getLength**(), untuk mengambil panjang datagram
- **getAddress**(), untuk mengambil alamat IP
- **getPort**(), untuk mengambil alamat port

Exception

- public class **SocketException** extends IOException
  - Kelas ini merupakan kelas yang diturunkan dari kelas IOException. Kelas exception ini dipanggil atau dipicu ketika ada kegagalan dalam pemakaian socket
  - contoh: kegagalan dalam protokol TCP.
    - ketika port yang akan digunakan sudah digunakan sebelumnya pada host tertentu

Exception (2)

- public class **BindException** extends SocketException
  - Kelas ini akan dipanggil ketika ada port lokal yang akan digunakan sudah terpakai oleh yang lain, atau ada kegagalan dalam permintaan untuk menggunakan alamat.
- public class **ConnectException** extends SocketException
  - Kelas ini akan dipanggil ketika sebuah koneksi ditolak oleh host yang dituju, oleh karena tidak ada proses yang siap menerima data pada port yang dituju.

Exception (3)

- public class **NoRouteToHostException** extends SocketException
  - Koneksi yang akan dibangun tidak dapat dipenuhi oleh karena melebihi waktu timeout yang tersedia atau host yang dituju tidak dapat dicapai (unreachable).
- public class **ProtocolException** extends IOException
  - Terjadi ketika data diterima dari network menyalahi aturan TCP/IP

Datagram Sockets

SERVER:

1. Create a DatagramSocket object
   ```
   DatagramSocket dgramSocket =
           new DatagramSocket(1234);
   ```
2. Create a buffer for incoming datagrams
   ```
   byte[] buffer = new byte[256];
   ```
3. Create a *DatagramPacket* object for the incoming datagram:
   - constructor get 2 arguments: the previously-created byte array; the size of this array.

   ```
   DatagramPacket inPacket = new
     DatagramPacket(buffer, buffer.length);
   ```
4. Accept an incoming datagram
   ```
   dgramSocket.receive(inPacket)
   ```

Datagram Sockets

SERVER:

5. Accept the sender's address and port from the packet
   ```
   InetAddress clientAddress = inPacket.getAddress();
   int clientPort = inPacket.getPort();
   ```
6. Retrieve the data from the buffer: 3 arguments: byte array, start byte array position, and length of byte array
   ```
   string message = new String(inPacket.getData(), 0,
   inPacket.getLength());
   ```
7. Create the response datagram. 3 arguments:

   the byte array containing the response message; the size of the response; the client's address; the client's port number.

   ```
   DatagramPacket outPacket = new
      DatagramPacket(response.getBytes(),
      response.length(),clientAddress, clientPort);
   ```
5. Send the response datagram
   ```
   dgramSocket.send(outPacket)
   ```
6. Close the *DatagramSocket:* `dgram.close();`

InfoServerUDP

```
1  import java.io.*;
2  import java.net.*;
3  import java.util.*;
4
5  public class InfoServerUDP {
6     private final int INFO_PORT=50000;
7     private string datafromClient;
8
9     /** InfoServerUDP Constructor */
10    public InfoServerUDP() {
11       DatagramSocket serverSocket;
12
13       try {
14          /* bind port 50000 ke alamat lokal */
15          serverSocket = new DatagramSocket(INFO_PORT);
16          System.out.println("Server telah siap...");
17
18          /* lakukan perulangan tanpa henti,
19           * agar layanan UDP ini dapat tersedia untuk
20           * koneksi satu client pada saat yang sama
21           */
22          while (true) {
23
24             /* lakukan perulangan sampai client
25              * mengirimkan perintah QUIT
26              */
27             boolean isQUIT = false;
28             while (!isQUIT) {
29                byte[] byteFromClient = new byte[1024];
30                byte[] byteToClient = new byte[1024];
31
32                DatagramPacket receivePacket =
33                   new DatagramPacket(
34                      byteFromClient, byteFromClient.length);
35
36                /* baca data dari client */
37                serverSocket.receive(receivePacket);
38
```

```
41              int port = receivePacket.getPort();
42
43              String data =
44                    new String(receivePacket.getData());
45
46              if (data.startsWith("TIME")) {
47                String DateNow =
48                     new String(new Date().toString());
49                byteToClient = DateNow.getBytes();
50
51              } else if (data.startsWith("NET")) {
52                String hostname = new String(
53             InetAddress.getByName("localhost").toString());
54                byteToClient = hostname.getBytes();
55
56              } else if (data.startsWith("QUIT")) {
57                isQUIT = true;
58                String thanks =
59                     new String("Terima kasih!");
60                byteToClient = thanks.getBytes();
61              }
62              DatagramPacket sendPacket =
63                new DatagramPacket(byteToClient,
64                   byteToClient.length, IPAddress, port);
65              serverSocket.send(sendPacket);
66            }
67
68            System.out.println("Hub. client tertutup..");
69          }
70        }
71        catch (IOException ioe) {
72          System.out.print("error: " + ioe);
73        }
74        catch (Exception e) {
75          System.out.print("error: " + e);
76        }
77    }
78
79    /* program utama */
80    public static void main(String[] args) {
81      new InfoServerUDP();
82    }
83  }
```

Datagram Sockets

CLIENT:

1. Create a DatagramSocket object
   `DatagramSocket dgramSocket = new DatagramSocket();`
2. Create the outgoing datagram
   `DatagramPacket outPacket = new DatagramPacket(message.getBytes(), message.length(),host, port);`
3. Send the datagram message
   `dgramSocket.send(outPacket)`
4. Create a buffer for incoming datagrams
   `byte[] buffer = new byte[256];`

Datagram Sockets

CLIENT:

5. Create a *DatagramPacket* object for the incoming datagram

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

6. Accept an incoming datagram

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

7. Retrieve the data from the buffer

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

8. Close the *DatagramSocket:*

```
dgram.close();
```

InfoClientUDP

```
 1 import java.net.*;
 2 import java.io.*;
 3 import java.util.*;
 4
 5 public class InfoClientUDP {
 6    private final int INFO_PORT=50000;
 7    private final String TargetHost = "localhost";
 8    private final String QUIT = "QUIT";
 9    private DatagramSocket clientSocket;
10
11    public InfoClientUDP() {
12       try {
13          /* siapkan input stream dari keyboard */
14          BufferedReader inFromUser =
15              new BufferedReader(
16                 new InputStreamReader(System.in));
17
18          /* buat koneksi datagram socket */
19          clientSocket = new DatagramSocket();
20
21          /* siapkan struktur untuk alamat IP
22           * yang dituju */
23          InetAddress IPAddress =
24              InetAddress.getByName("localhost");
25
26          boolean isQuit = false;
27          while (!isQuit) {
28             /* siapkan buffer untuk data yang diterima
29              * atau dikirim */
30             byte[] byteFromServer = new byte[1024];
31             byte[] byteToServer = new byte[1024];
```

```
33              /* menunggu masukan perintah dari User */
34              System.out.print("Perintah Anda : ");
35              String cmd = inFromUser.readLine();
36
37              /* konversi ke Upper dan cek
38               * apakah perintah QUIT yang diketikkan
39               * jika QUIT, beri nilai true untuk isQUIT
40               * agar looping !isQUIT selesai
41               */
42              cmd = cmd.toUpperCase();
43              isQuit = cmd.equals(QUIT);
44              byteToServer = cmd.getBytes();
45
46              DatagramPacket sendPacket =
47                  new DatagramPacket(byteToServer,
48                  byteToServer.length, IPAddress, INFO_PORT);
49              clientSocket.send(sendPacket);
50
51              /* block reading ....
52               * client harus menunggu balasan dari server
53               */
54              DatagramPacket receivePacket =
55                new DatagramPacket(byteFromServer,
56                                   byteFromServer.length);
57
58              clientSocket.receive(receivePacket);
59              String result =
60                  new String(receivePacket.getData());
61              System.out.println("Dari Server: " + result);
62          }
63
64          /* tutup semua stream dan koneksi socket */
65          clientSocket.close();
66      }
67      catch (IOException ioe) {
68          System.out.println("Error:" + ioe);
69      }
70      catch (Exception e) {
71          System.out.println("Error:" + e);
72      }
73  }
74
```

```
75    /* program utama */
76    public static void main(String[] args) {
77      new InfoClientUDP();
78    }
79  }
```

Prinsip-prinsip yang dilakukan oleh InfoClientUDP

- Buat DatagramSocket
- Lakukan loop sampai user mengetikkan QUIT.
  - Baca masukkan dari user
  - Tampung pada buffer array byte
  - Buat obyek DatagramPacket untuk dikirimkan ke server
  - Kirimkan DatagramPacket ke server
  - Siapkan packet datagram untuk mengambil informasi dari client
  - Baca DatagramPacket yang dikirim dari client
- Setelah Client QUIT, tutup DatagramSocket.

NEXT

- Distributed Object & Remote Invocation
  - Chapter 5
  - RPC, RMI & CORBA