What is Networking?

Networking is the process of connecting two or more computing devices together so that data and resources can be shared.

Networking Terminologies

The widely used networking terminologies are given below:

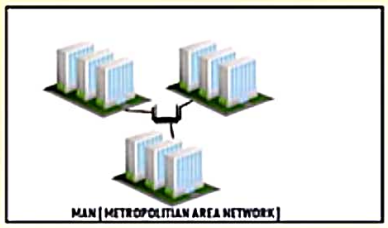
1. *LAN-MAN-WAN*
2. *WWW*
3. *IP Address*
4. *Port Number*
5. *URL*
6. *MAC Address*
7. *Socket*

LAN-MAN-WAN

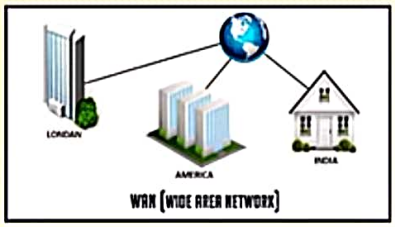
A **LAN** (Local Area Network) is a group of computers and network devices connected together, usually within the same building/ campus.

By definition, the connections must be high speed and relatively inexpensive (e.g., token ring or Ethernet). Most University departments are on LANs.

A **MAN** (Metropolitan Area Network) is a larger network that usually spans several buildings in the same city or town. Cable TV and Cable Internet are good example for MAN.



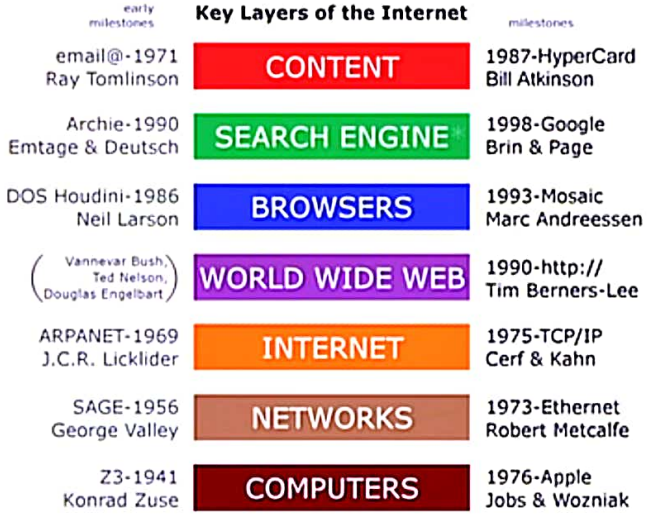
A **WAN** (Wide Area Network), in comparison to a MAN, is not restricted to a geographical location, although it might be confined within the bounds of a state or country. The technology is high speed and relatively expensive. The Internet is an example of a worldwide public WAN.



WWW

The World Wide Web (WWW), also called the Web and W3, is an information space where documents and other web resources are identified by Uniform Resource Locators (URLs), interlinked by hypertext links, and accessible via the Internet.

English scientist Tim Berners-Lee invented the World Wide Web in 1989.

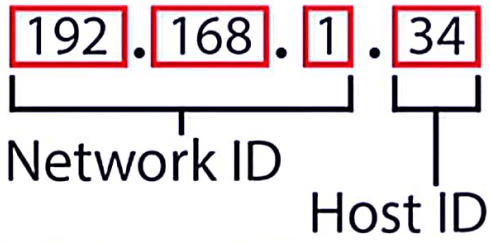


IP Address

IP address is a unique number assigned to a node of a network, for example, 192.168.0.1 .

It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

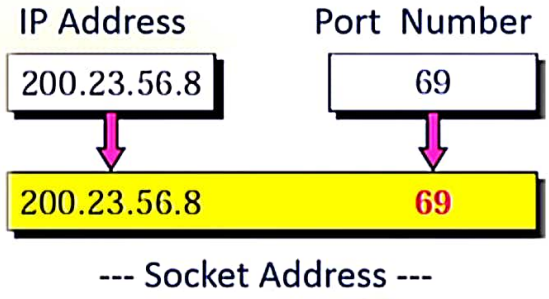


Port number

The port number is used to uniquely identify different applications.

It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

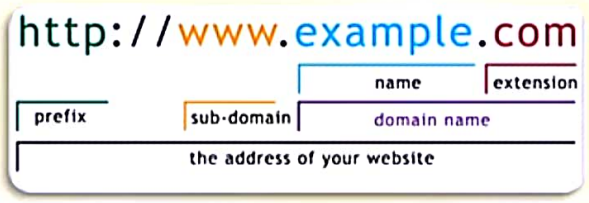


URL

URL is the abbreviation of **Uniform Resource Locator** and is defined as the global address of documents and other resources on the World Wide Web.

To visit some website, for example Google, you'll go to the URL www.google.com.

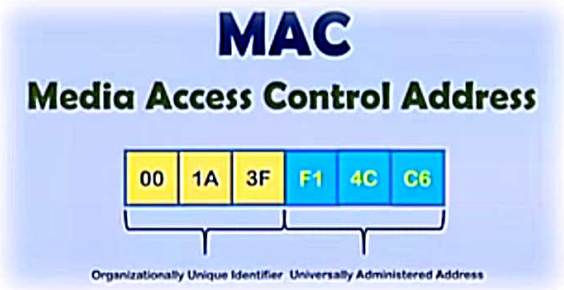
- The first part of the URL is called *protocol identifier* and it indicates what protocol to use,
- Second part is called *resource name* and it specifies the IP address or the domain name where the resource is located.
- The protocol identifier and the resource name are separated by a *colon* and *two forward slashes.*



MAC address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller).

A network node can have multiple NICs but each with unique MAC.



Socket

A socket is an application program responsible for communication between two end points.
A socket is uniquely identified by an IP address and a Port.

Note: A socket is a software element, and it does not imply any hardware.

Communication Protocols

1. Connection-less protocol *(UDP – User Datagram Protocol)*
2. Connection-oriented protocol *(TCP – Transmission Control Protocol)*
3. TCP-IP *(Internet Protocol)*
4. FTP *(File Transfer Protocol)*
5. HTTP *(Hypertext Transfer Protocol)*
6. HTTPS *(Secure Hypertext Transfer Protocol)*
7. SMTP *(Simple Mail Transfer Protocol)*

Connection-oriented protocol

In connection-oriented protocol, acknowledgement is sent by the receiver.

So it is reliable but slow.

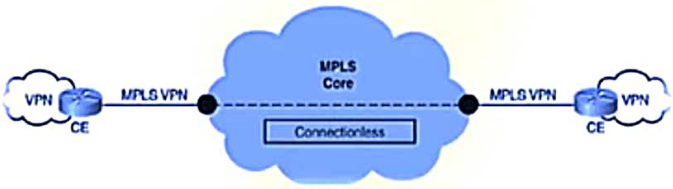
TCP follows connection-oriented protocol.

Connection-less protocol

In connection-less protocol, acknowledgement is not sent by the receiver.

So it is not reliable but fast.

The example of connection-less protocol is UDP.



TCP-IP

Transmission Control Protocol (TCP) – It is known to provide reliable and error-free communication between end systems.

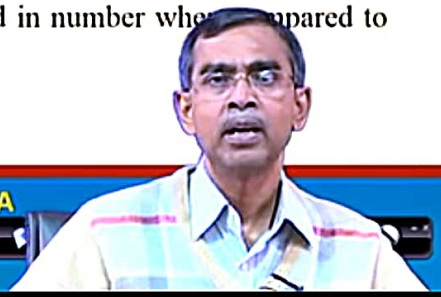
- It performs sequencing and segmentation of data.
- It also has acknowledgment feature and controls the flow of the data through flow control mechanism.
- It is a very effective protocol but has a lot of overhead due to such features. Increased overhead leads to increased cost.

Internet Protocol (IP) – It is responsible for delivering packets from the source host to the destination host by looking at the IP addresses in the packet headers.

IP has 2 versions: IPv4 and IPv6.

IPv4 is the one that most of the websites are using currently.

But IPv6 is growing as the number of IPv4 addresses are limited in number when compared to the number of users.



FTP

File Transfer Protocol (FTP) is an application layer protocol which moves files between local and remote file systems. It runs on the top of TCP, like HTTP. To transfer a file, 2 TCP connections are used by FTP in parallel: *control connection and data connection.*

Control connection

For sending control information like user identification, password, commands to change the remote directory, commands to retrieve and store files etc., FTP makes use of control connection. Control connection is initiated on port number 21.

Data connection

For sending the actual file, FTP makes use of data connection. Data connection is initiated on port number 20.

FTP sends the control information out-of-band as it uses a separate control connection. Some protocols send their request and response header lines and the data in the same TCP connection. For this reason, they are said to send their control information in-band. HTTP and SMTP are such examples.

HTTP & HTTPS

HTTP means Hyper Text Transfer Protocol.

HTTP is the underlying protocol used by the World Wide Web

This protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

HTTPS means Hyper Text Transfer Protocol Secure

It is the secure version of HTTP

It means all communications between your browser and the website are encrypted.

SMTP

SMTP means Simple Mail Transfer Protocol

SMTP is an application layer protocol.

The client who wants to send the mail opens a TCP connection to the SMTP server and then sends the mail across the connection. The SMTP server is always on listening mode. As soon as it listens for a TCP connection from any client, the SMTP process initiates a connection on that port (numbered as 25. After successfully establishing the TCP connection the client process sends the mail instantly.

Overview HTTP-FTP-SMTP

PARAMETER	HTTP	FTP	SMTP
Port number	80	20 and 21	25
Type of band transfer	In-band	Out-of-band	In-band
State	Stateless	Maintains state	–
Number of TCP connections	1	2 (Data Connection and Control Connection)	1
Type of TCP connection	Can use both Persistent and Non-persistent	Persistent for Control connection. Non-persistent for Data Connection	Persistent
Type of Protocol	Pull Protocol (Mainly)	–	Push Protocol (Primarily)
Type of Transfer	Transfer files between Web server and Web client	Transfer directly between computers	Transfers mails via Mail Servers

Objectives

For the networking with Java, it provides two types of sockets: stream sockets and datagram sockets.

A. Stream Sockets

1. With stream sockets a process establishes a connection to another process.
2. While the connection is in place, data flows between the processes are continuous streams.
3. Here, stream sockets are said to provide a connection-oriented service.
4. The protocol used is TCP (Transmission Control Protocol).

B. Datagram Sockets

1. With datagram sockets, individual packets of information are transmitted.
2. The transmission of packets follows a connection less service.
3. The protocol used is UDP (User Datagram Protocol).

TCP versus UDP

TRANSMISSION CONTROL PROTOCOL (TCP)	USER DATAGRAM PROTOCOL (UDP)
TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission.
TCP is reliable as it guarantees delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error checking mechanism using checksums.
Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver.	There is no sequencing of data in UDP. If ordering is required, it has to be managed by the application layer.
TCP is comparatively slower than UDP.	UDP is faster, simpler and more efficient than TCP.
Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in User Datagram Protocol (UDP).
TCP header size is 20 bytes. TCP is heavy-weight.	UDP Header size is 8 bytes. UDP is lightweight.
TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.

Classis in Java for Networking

URL

URLConnection

HttpURLConnection

InetAddress

DatagramSocket

ServerSocket

Socket

java.net package

Class URL

https://nptel.ac.in/course.php

A URL contains many information. For example

- Protocol: In this case, https is the protocol.
- Server name or IP Address: In this case, nptel.ac.in is the server name.
- Port Number: It is an optional attribute. If we write https://nptel.ac.in:80/ , 80 is the port number. If port number is not mentioned in the URL, it will return **-1**.
- File Name or directory name: In this case, course.php is the file name.

Class URL : Methods

The `java.net.URL` class provides many methods. The important methods of URL class are given below.

Method	Description
`public String getProtocol()`	Returns the protocol of the URL.
`public String getHost()`	Returns the host name of the URL.
`public String getPort()`	Returns the Port Number of the URL.
`public String getFile()`	Returns the file name of the URL.
`public URLConnection openConnection()`	It returns an instance of URLConnection associated with th

Class URL : An example

```
import java.net.*;

public class URLClass{
     public static void main(String[] args){
          try{
             URL url = new URL("https://nptel.ac.in/course.php");

             System.out.println("Protocol: "+url.getProtocol());
             System.out.println("Host Name: "+url.getHost());
             System.out.println("Port Number: "+url.getPort());
             System.out.println("File Name: "+url.getFile());

          } catch(Exception e){
             System.out.println(e);
          }
     }
}
```

Class URLConnection

The Java `URLConnection` class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

Some of the important **methods** provided by this class is shown below:

Method	Description
`openConnection()`	Returns the object of URLConnection class.
`getInputStream()`	Returns all the data of the specified URL in the stream that can be read and displayed

The syntax to get the object of URLConnection class

```
public URLConnection openConnection() throws IOException{}
```

Class URLConnection : An example

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

public class URLConnectionClass{
    public static void main(String[] args){
       try{
              URL url = new URL("https://nptel.ac.in/course.php");

              URLConnection urlcon = url.openConnection();

              InputStream stream = urlcon.getInputStream();

              int b;
              while((b = stream.read()) != -1){
                  System.out.print((char)b);
              }
              }catch(Exception e){
                  System.out.println(e);}
     }
}
```

Class HttpURLConnection

The Java `HttpURLConnection` class is http specific `URLConnection`. It works for **HTTP protocol** only.

This class can be used for extracting the following information of any HTTP URL

- Header information.
- Status code.
- Response code, etc.

The syntax to get the object of **HttpURLConnection** class using typecasting

```
1. public URLConnection openConnection() throws IOException{}

2. URL url = new URL("http://www.nptel.ac.in/java-tutorial");

3. HttpURLConnection huc = (HttpURLConnection) url.openConnection();
```

Class HttpURLConnection

The Java `URLConnection` class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

Some of the important **methods** provided by this class is shown below:

Method	Description
getHeaderFieldKey(int n)	Returns the information in the header field n.
getHeaderField(java.lang.String name)	Returns all the header field.

Note: There are many more methods in this class

The syntax to get the object of URLConnection class

```
public URLConnection openConnection() throws IOException{}
```

Class HttpURLConnection : An example

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

public class HttpURLConnectionDemo{
   public static void main(String[] args){
      try{
          URL url = new URL("https://nptel.ac.in/course.php");
          HttpURLConnection huc = (HttpURLConnection)url.openConnection();
          for(int i=1;i<=8;i++){
             System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
          }
          huc.disconnect();
       }catch(Exception e){
          System.out.println(e);
       }
   }
}
```

Class InetAddress

Java **InetAddress** class represents an IP address. The **java.net.InetAddress** class provides methods to get the IP of any host name, For examples:

www.nptel.ac.in www.google.com www.wikipedia.org

The **java.net.InetAddress** class provides many methods. The important methods of this class are given below.

Method	Description
`public static InetAddress getByName(String host) throws UnknownHostException`	Returns the instance of InetAddress containing LocalHost IP and name.
`public static InetAddress getLocalHost() throws UnknownHostException`	Returns the instance of InetAdddress containing local host name and address.
`public String getHostName()`	Returns the host name of the IP address.
`public String getHostAddress()`	Returns the IP address in string format.
`public URLConnection openConnection()`	It returns instance of URLConnection i.e. associated with this URL.

Class InetAddress : An example

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

public class InetAddressClass{
     public static void main(String[] args){
          try{
               InetAddress ip = InetAddress.getByName("www.nptel.ac.in");

               System.out.println("Host Name: "+ip.getHostName());
               System.out.println("IP Address: "+ip.getHostAddress());

          }catch(Exception e){
               System.out.println(e); }
     }
}
```

Class DatagramSocket

Java `DatagramSocket` class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class:

Constructors	Description
`DatagramSocket() throws SocketException`	It creates a datagram socket and binds it with the available Port Number on the localhost machine.
`DatagramSocket(int port) throws SocketException`	It creates a datagram socket and binds it with the given Port Number.
`DatagramSocket(int port, InetAddress address) throws SocketException`	It creates a datagram socket and binds it with the specified port number and host address.

Class DatagramSocket : Methods

The `DatagramSocket` class provides many methods. The important methods of this class are given below.

Method	Description
`public InetAddress getAddress ( )`	Returns the destination InetAddress. It is typically used for sending.
`public int getport ( )`	Returns the integer destination port number. It is typically used for sending.
`public byte [ ] getData ( )`	Returns the byte array of data contained in the datagram. It is used to retrieve data from the datagram after it has been received.
`public int getLength ( )`	Returns the length of the valid data contained in the byte array that would be returned from the getData() method.

Class DatagramSocket : An example

```
import java.net.*;
public class Receiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Receiver

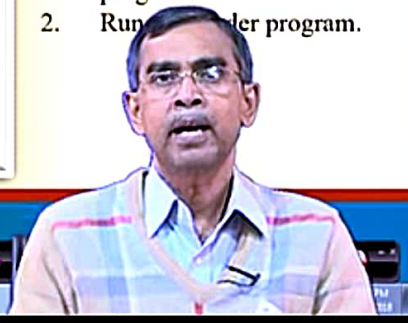
```
import java.net.*;
public class Sender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome to NPTEL";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

Sender

This example illustrate how a sender program sends some messages to the receiver program using **DatagramSocket** UDP protocol.

Steps:

1. First run the receiver program.
2. Run [illegible]der program.



Class ServerSocket

The `java.net.ServerSocket` class is used by **server** applications to obtain a *port* and *listen* for client requests.

`ServerSocket` class has the following *constructors*:

Constructors	Description
`public ServerSocket(int port) throws IOException`	Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
`public ServerSocket(int port, int backlog) throws IOException`	Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
`public ServerSocket(int port, int backlog, InetAddress address) throws IOException`	Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addr… accept client requests on.
`public ServerSocket() throws IOException`	Creates an unbound server socket. When using this constr… ind() method when you are ready to bind the server socket.

Class ServerSocket

The `ServerSocket` class can be used to create a server socket. This object is used to establish communication with the clients and it has the following methods:

Method	Description
`public int getLocalPort()`	It connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
`public Socket accept() throws IOException`	This method accepts a connection to another socket (client).
`public void setSoTimeout(int timeout)`	Connects to the specified host and port, creating a socket on the local host at the specified address and port.
`public void bind(SocketAddress host, int backlog)`	This method establishes a connection for a certain time.

Class Socket

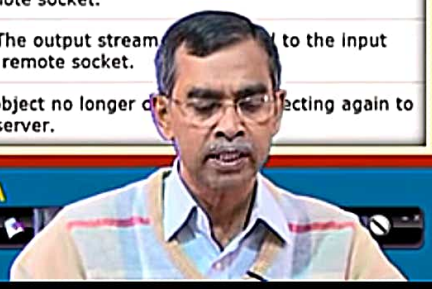
A socket is simply an endpoint for communications between the machines. The `Socket` class can be used to create a socket. It has the following constructors :

Constructors	Description
`public Socket(String host, int port) throws UnknownHostException, IOException`	It connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server
`public Socket(InetAddress host, int port) throws IOException`	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object
`public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException`	Connects to the specified host and port, creating a socket on the local host at the specified address and port
`public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException`	This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String
`public Socket()`	Creates an unconnected socket. Use the connect() method to connect this socket to a server.

Class Socket

Following are the methods present in the `Socket` class.:

Method	Description
`public void connect(SocketAddress host, int timeout) throws IOException)`	This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
`public InetAddress getInetAddress()`	This method returns the address of the other computer that this socket is connected to.
`public int getPort()`	Returns the port the socket is bound to on the remote machine.
`public int getLocalPort()`	Returns the port the socket is bound to on the local machine.
`public SocketAddress getRemoteSocketAddress()`	Returns the address of the remote socket.
`public InputStream getInputStream() throws IOException`	Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
`public OutputStream getOutputStream() throws IOException`	Returns the output stream of the socket. The output stream ... to the input stream of the remote socket.
`public void close() throws IOException`	Closes the socket, which makes this Socket object no longer ... ecting again to any server.



Example 1 : Simple Client-Server communication

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

public class MyServer {
 public static void main(String[] args){
  try{
   ServerSocket ss = new ServerSocket(6666);
   Socket s = ss.accept();//establishes connection
   DataInputStream dis = new DataInputStream(s.getInputStream());
   String  str = (String)dis.readUTF();
   System.out.println("message= "+str);
   ss.close();
   }catch(Exception e){
     System.out.println(e);
   }
 }
}
```

Server

Let's see a simple of java socket programming in which client sends a text and server receives it. Shows the message and then closes the connection and execution stops.

This is the **server program**.

Example 1 : Simple Client-Server Communication

This is the client program.

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

public class MyClient {
public static void main(String[] args) {
   try{
      Socket s = new Socket("localhost",6666);
      DataOutputStream dout = new DataOutputStream(s.getOutputStream());
      dout.writeUTF("Hello Server");
      dout.flush();
      dout.close();
      s.close();
    }catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Client

Example 2 : Dialogue Client-Server

```
import java.net.*;
import java.io.*;
class MyServer{
 public static void main(String args[])throws Exception{
     ServerSocket ss = new ServerSocket(3333);
     Socket s = ss.accept();
  DataInputStream din = new DataInputStream(s.getInputStream());
  DataOutputStream dout = new DataOutputStream(s.getOutputStream());
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  String str1 = "",str2= "";
  while(!str1.equals("stop")){
   str1 = din.readUTF();
   System.out.println("client says: "+str);
   str2 = br.readLine();
   dout.writeUTF(str2);
   dout.flush();
  }
  din.close();
  s.close();
  ss.close();
 }
}
```

Server

In this example, client will write first to the server then server will receive and print the text.

Then server will write to the client and client will receive and print the text. The step goes on.

This is the **server program** and next slide contains the **client program**.

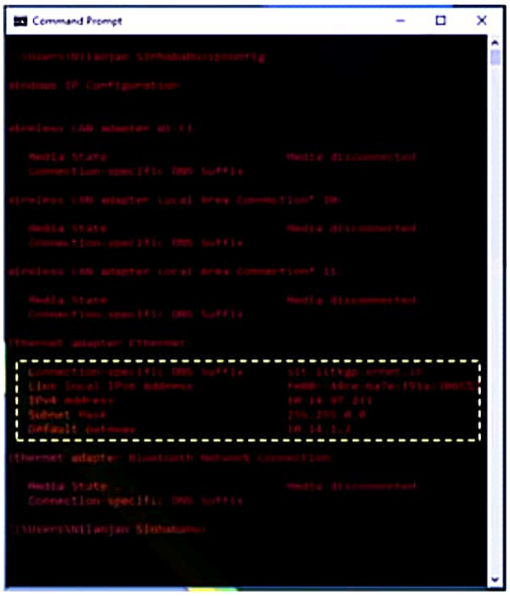
Example 2 : Dialogue Client-Server

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str1="",str2="";
while(!str1.equals("stop")){
Str1 = br.readLine();
dout.writeUTF(str1);
dout.flush();
Str2 = din.readUTF();
System.out.println("Server says: "+str2);
}
dout.close();
s.close();
}}
```

Client

This is the **client program.**

Example 3 : Remote Client-Server dialogue



This is the same example as the previous program. The only difference is that *the server and the client are on different machines*.

In this case, the only difference is that the server should listen to the LAN IP address instead of local host and the client should know the IP and port of the server

This is the **server program** and next slide contains the **client program**.

Example 3 : Remote Client-Server dialogue

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

class MyServer{
  public static void main(String args[])throws Exception{
    ServerSocket ss = new ServerSocket(3333,2,InetAddress.getByName("10.14.97.211"));
    Socket s = ss.accept();
    DataInputStream din=new DataInputStream(s.getInputStream());
    DataOutputStream dout=new DataOutputStream(s.getOutputStream());
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String str1 ="",str2="";
      while(!str1.equals("stop")){
        str1 = din.readUTF();
        System.out.println("client says: "+str1);
               str2 = br.readLine();
               dout.writeUTF(str2);
               dout.flush();
             }
    din.close();
    s.close();
  }
}
```

Server

This is the same example as the previous program. The only difference is that *the server and the client are on different machines*.

In this case, the only difference is that the server should listen to the LAN IP address instead of local host and the client should know the IP and port of the server

This is the **server program** and next slide contains the **client program**.

Example 3 : Remote Client-Server dialogue

```
import java.net.*;
import java.io.*;
class MyClient{
     public static void main(String args[])throws Exception{
          Socket s = new Socket("10.14.97.211",3333);
          DataInputStream din=new DataInputStream(s.getInputStream());
          DataOutputStream dout=new DataOutputStream(s.getOutputStream());
          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
          String str1 = "",str2 = "";
          while(!str.equals("stop")){
               str=br.readLine();
               dout.writeUTF(str);
               dout.flush();
               str2 = din.readUTF();
               System.out.println("Server says: "+str2);
           }
           dout.close();
           s.close();
     }
}
```

Client

This is the **client program.**

Concurrent server : An example

```
import java.io.*;
import java.text.*;
import java.util.*;
import java.net.*;
// Server class
public class MyServer{
    public static void main(String[] args) throws IOException{
        // server is listening on port 5056
        ServerSocket ss = new ServerSocket(5056);
        // running infinite loop for getting
        // client request
        while (true){
            Socket s = null;
            try{
                // socket object to receive incoming client requests
                s = ss.accept();
                System.out.println("A new client is connected : " + s);
                // obtaining input and out streams
                DataInputStream dis = new DataInputStream(s.getInputStream());
                DataOutputStream dos = new DataOutputStream(s.getOutputStream());
                System.out.println("Assigning new thread for this client");
                // create a new thread object
                Thread t = new ClientHandler(s, dis, dos);
                // Invoking the start() method
                t.start();
            )
```

Server

In this example, the we will create a Date-Time server, and clients can either view '***Date***' or '***Time***' as per their selection. Also, Clients can close the connection by typing '***Exit***'.

Since we will be using threads in the program, multiple clients can be connected and request information from the server symultaniously.

This is the **server program** followed by the **client program**.

Example 3 : Concurrent server

```
 catch (Exception e){
                s.close();
                e.printStackTrace();
            }
        }
    }
}
// ClientHandler class
class ClientHandler extends Thread
{
    DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
    DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
    final DataInputStream dis;
    final DataOutputStream dos;
    final Socket s;
    // Constructor
    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos){
        this.s = s;
        this.dis = dis;
        this.dos = dos;
    }

    public void run(){
        String received;
        String toreturn;
```

Server

This is the **server program.**

Example 3 : Concurrent server

```
while (true){
        try {
            // Ask user what he wants
            dos.writeUTF("What do you want?[Date | Time]..\n"+
            "Type Exit to terminate connection.");
            // receive the answer from client
            received = dis.readUTF();
            if(received.equals("Exit")){
                System.out.println("Client " + this.s + " sends exit...");
                System.out.println("Closing this connection.");
                this.s.close();
                System.out.println("Connection closed");
                break;
            }
            // creating Date object
            Date date = new Date();
            // write on output stream based on the
            // answer from the client
            switch (received) {
                case "Date" :
                    toreturn = fordate.format(date);
                    dos.writeUTF(toreturn);
                    break;
```

Server

This is the **server program.**

Example 3 : Concurrent server

```
                    case "Time" :
                        toreturn = fortime.format(date);
                        dos.writeUTF(toreturn);
                        break;
                    default:
                        dos.writeUTF("Invalid input");
                        break;
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        try{
            // closing resources
            this.dis.close();
            this.dos.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Server

This is the **server program.**

Example 3 : Concurrent server (Client)

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
// Client class
public class MyClient{
    public static void main(String[] args) throws IOException{
        try{
            Scanner scn = new Scanner(System.in);
            // getting localhost ip
            InetAddress ip = InetAddress.getByName("localhost");
            // establish the connection with server port 5056
            Socket s = new Socket(ip, 5056);
            // obtaining input and out streams
            DataInputStream dis = new DataInputStream(s.getInputStream());
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());
            // the following loop performs the exchange of
            // information between client and client handler
            while (true){
                System.out.println(dis.readUTF());
                String tosend = scn.nextLine();
                dos.writeUTF(tosend);
```

Client

This is the **client program.**