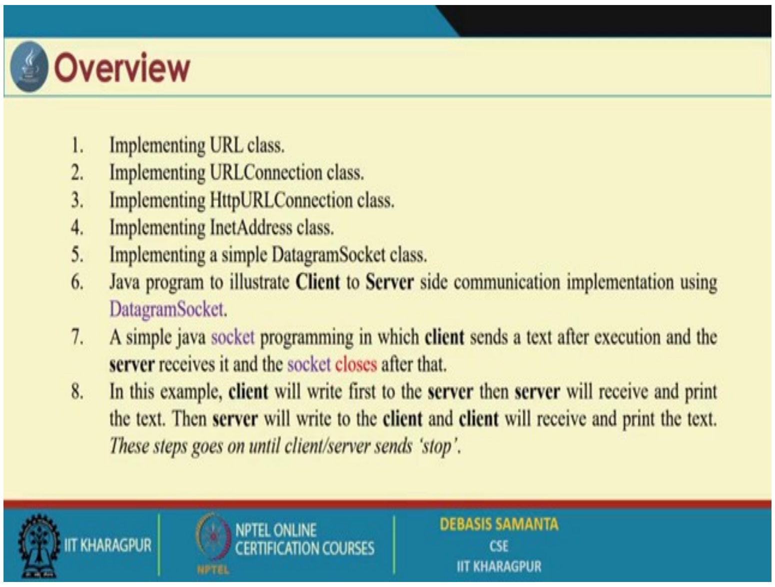
Programming in JAVA
Prof. Debasis Samanta
Department of Compute Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 49
Demonstration – XIX

Now, we have quick demonstrations of the different classes that we have discussed in our last module related to the Java networking and as you have learned about the different classes are there.

(Refer Slide Time: 00:31)



Overview

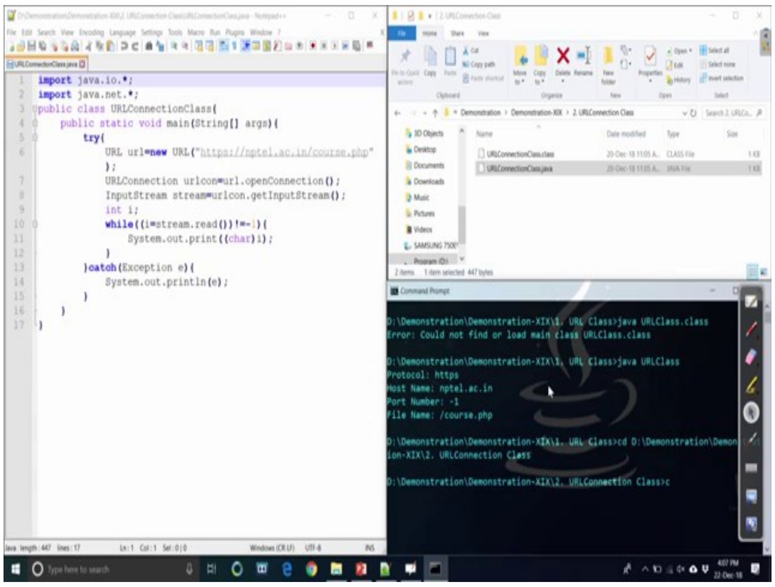
1. Implementing URL class.
2. Implementing URLConnection class.
3. Implementing HttpURLConnection class.
4. Implementing InetAddress class.
5. Implementing a simple DatagramSocket class.
6. Java program to illustrate **Client** to **Server** side communication implementation using DatagramSocket.
7. A simple java socket programming in which **client** sends a text after execution and the **server** receives it and the socket closes after that.
8. 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. *These steps goes on until client/server sends 'stop'.*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE IIT KHARAGPUR

So, in this Demonstration we will try to emphasize about the different classes and their utilizations and then how the internet how the URL related information that can be accessed given the URL and like this one. And then mainly we will discuss about the client server; that means, how the 2 sockets can be designed and then that 2 sockets can communicate to each others.

And there may be one server sockets and the client sockets like and they are also the two way communications also and then there a concurrent communication; that means, ones client one server can serve the request to many clients which are connected to that server. So, those things we will try to give an I mean demonstration in this session. So, that let us have the first program and this program is related to the URL class utilizations as we have said that every URL can be created as an object. So, this is called the URL objects.

(Refer Slide Time: 01:29)



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

```
D:\Demonstration\Demonstration-XIX\1. URL Class>java URLClass.class
Error: Could not find or load main class URLClass.class

D:\Demonstration\Demonstration-XIX\1. URL Class>java URLClass
Protocol: https
Host Name: nptel.ac.in
Port Number: -1
File Name: /course.php

D:\Demonstration\Demonstration-XIX\1. URL Class>cd D:\Demonstration\Demonstrat
ion-XIX\2. URLConnection Class

D:\Demonstration\Demonstration-XIX\2. URLConnection Class>c
```

So, for example, https://nptel.ac.in/ some file name is basically is the URL and then regarding this URL we can make a connection. So, the connection is that whatever the location we have specified to that the distance machine or local host whatever be that, so, it will make an connection. But here URL basically in other than connection we can have the many methods are there by which you can access the information about the url. For example, which Protocol that it has been used or what is the Name of the server, what is the ip address of the server, what is the Port Number, what is the File that we are going to access whatever it is there.

So, all these informations can be accessed here. As you see in this program we create an url object. So, this is basically url object here and then we just the is the constructor of the URLClass argument is the URL specification this is the url actually for which you want to have the information. And then this url is an object and for the subject we can call the different method like say getProtocol, then getHost, getPort num Port, then getFile everything. So, here basically file is this one and whatever it is there.

Now if we run this then definitely it will try to find a communication connection from this machine, this program actually. So, this program to this one server and if there is no I mean violation in the Protocol then definitely the connection will be successful. However, https is a secured protocol so, from every program it may not allow the transmission. Now let us see exactly if we try to run this program whether our https protocol service can allow you to access to this connection and then finally, fetching the file it is here. So, I will just run this program here.

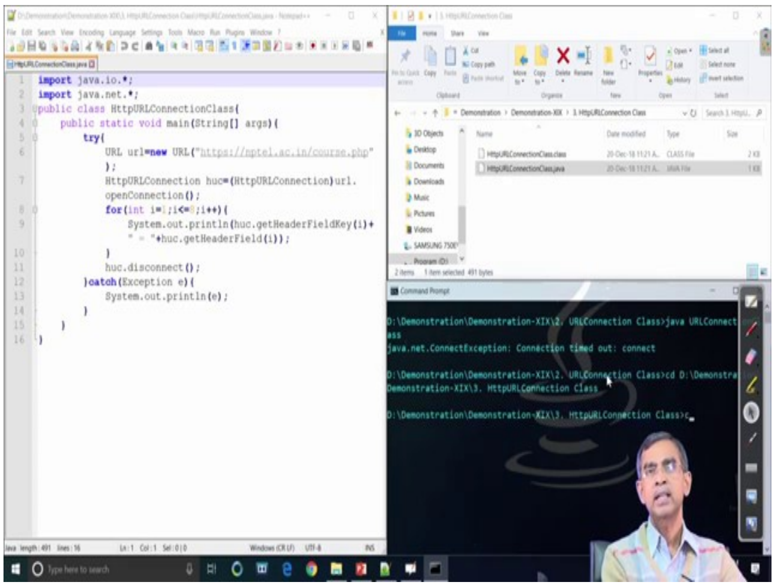
So, this program has been successfully compiled. So, no problem so far the compilation is concerned there. Anyway we are not getting the connection here, so, only we are getting the information about the crl. As you see the output; as you see the output here and we see that Protocol that is the https the HostName is basically nptel dot ac in and the PortNumber in this case is minus 1 and then file that basically which this URL specify course dot php is everything is a accepted from this method only ok.

So, this is the first example that and in major program actually we will I can have this URL class if it is then lot of information about these things can be accessed and then those information can be used for our communication purpose. Our second example is basically the URL connection; that means, it basically used to make a connection from the specified url location to the current program. And, this program if you see here again we create the url object here and then this is the url and then url connection is basically this create and object which basically a connection object here in this case and open url dot openConnection(); is basically for opening a correction.

And then once the connection is open, so, this basically stream it basically is a basically TCP protocol you can stream sockets like is a continuous stream can be stored here and this is the stream buffer we can use it here. So, that all the steam that can be connected by means of reading the distance machine the things will be there and here is basically displaying the content that we have there.

Now, here we have to establish the connection, now you just running the program and let us see after the execution whether we can get the file from the distance machine yes or not. Now here is the problem is because we are not able to establish the connection as this is the https actually, other than https or some other protocol if it can followed. Then it may be possible to face the file here in the local machine, but here because the protocol is a restriction, so, that all connections.

(Refer Slide Time: 05:39)



```
import java.io.*;
import java.net.*;
public class HttpURLConnectionClass{
    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);
        }
    }
}
```

```
D:\Demonstration\Demonstration-XIX\2. URLConnection Class>java URLConnect
ass
java.net.ConnectException: Connection timed out: connect

D:\Demonstration\Demonstration-XIX\2. URLConnection Class>cd D:\Demonstra
Demonstration-XIX\3. HttpURLConnection Class

D:\Demonstration\Demonstration-XIX\3. HttpURLConnection Class>c
```

We have sent the request; however, the protocol that implemented in the server end where is the nptel dot is that the refuse our connection, so, we could not get this one. So, here you can see connection timed out that mean they it has tried to several times and send the request, but that server has refused the connection that is why it is there.

So, because of this protocol only we could not could get it there, some other some of the machines other point if it is connected on it if we do it then maybe the communication is successful ok. So, this is the example regarding the URLClass and then URLConnection Class. Now here is an another example HttpURLConnection is basically the connection which basically follows the http protocol and this example is basically the similar to the previous one. And as you see that this is the url object we have created specifying the url location and then this is basically the opening the connection following the HttpURLConnection.

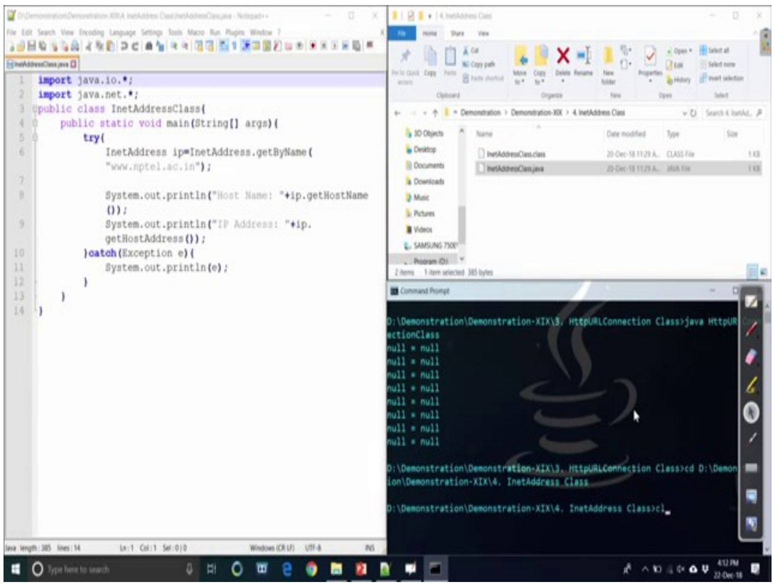
So, here is the object that we have followed to do that; that means, connection objects. And then for the entire connection if it is successful we read the entire document by just here actually we are not adding the entire document rather for this http connection actually http has the lot of information these are called the header information. So, HeaderField usually there are 8 fields are there.

So, in this program we attempted to read all the HeaderFields that is there in this connection objects. So, this program will do these things and if we run it then

HttpURLConnection, then we will be able to find the header information those are there in this connection right.

Here also because of the https we are not able to get any information regarding these what is the header fields and others. So, that is why the connection is not successful and then connection is timed out. Now, next example that we are going to discuss about communication using user datagram protocol which is the connection lists on service basically.

(Refer Slide Time: 07:49)



```java
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);
        }
    }
}
```

```
D:\Demonstration\Demonstration-XIX\3. HttpURLConnection Class>java HttpUR
ectionClass
null = null
null = null
null = null
null = null
null = null
null = null
null = null
null = null

D:\Demonstration\Demonstration-XIX\3. HttpURLConnection Class>cd D:\Demon
ion\Demonstration-XIX\4. InetAddress Class

D:\Demonstration\Demonstration-XIX\4. InetAddress Class>cl
```

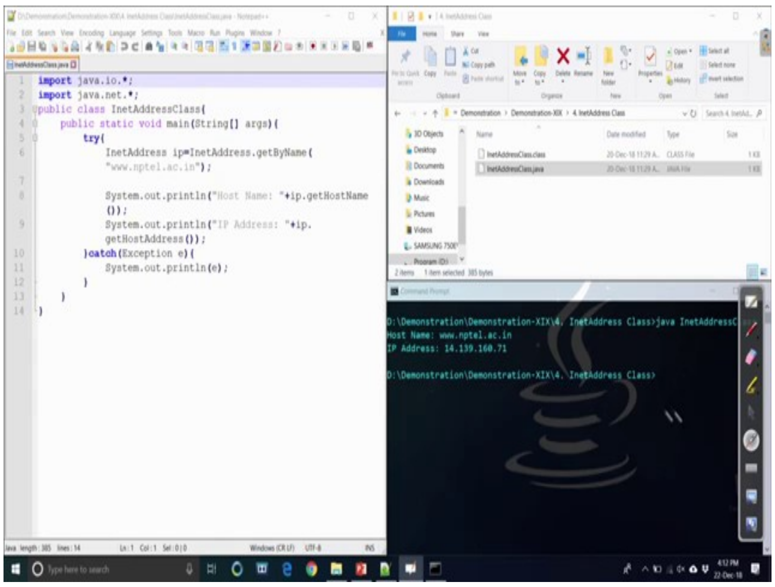
And before going to this InetAddressClass; so, this example related to the InetAddressClass and ok. So, InetAddress is basically the similar to the concept it is there InetAddress ip InetAddress guest getByName and this is basically the name of the ip locations or other is a server where it is located this by this is again a url sort of thing only by the domain name at that we have followed here. So, actually Inet is basically corresponding to this server as we have already mentioned there is a particular IPAddress, logical address.

So, basically with this ip is an object by which we will be able to access the different information regarding the inet addressing scheme. So, here so, getHostAddress what is the name of the host? What is the PortNumber? What is the name IPAddress everything we can be obtained here. For example, in this case we have to we are printing the IP

Address ip. So, IP Address like 8 bit 4 8 for 8 bit content in the ip for concept it will be displayed here.

Now let us run this program. So, that we will be able to see exactly what is the IP Address of www dot nptel dot ac dot in from this and then the guest hood address also yeah.

(Refer Slide Time: 09:11)



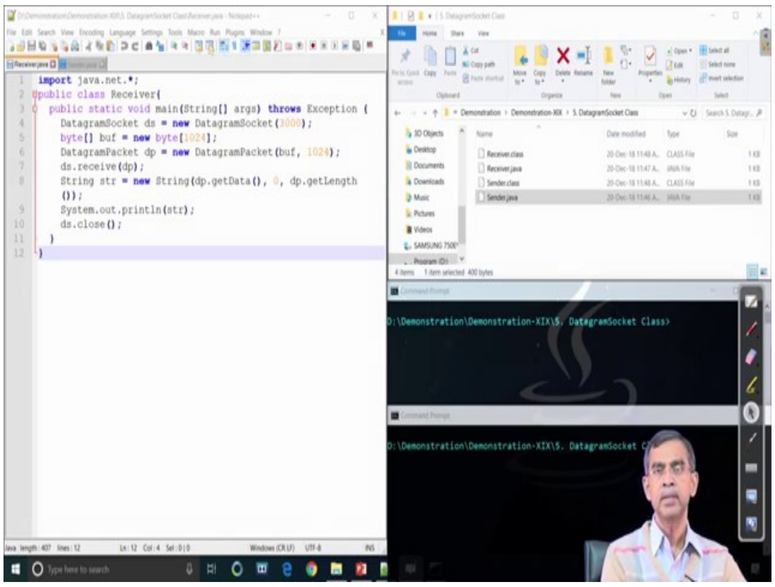
```java
import java.io.*;
import java.net.*;
public class InetAddressClass{
    public static void main(String[] args){
        try{
            InetAddress ip=InetAddress.getByName(
            "www.nptel.ac.in");

            System.out.println("Host Name: "+ip.getHostName
            ());
            System.out.println("IP Address: "+ip.
            getHostAddress());
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

```
D:\Demonstration\Demonstration-XIX\4. InetAddress Class>java InetAddressC
Host Name: www.nptel.ac.in
IP Address: 14.139.160.71

D:\Demonstration\Demonstration-XIX\4. InetAddress Class>
```

As you see here the Host Name is www.nptel.ac.in and then IP Address for this particular machine is 14 dot 139 dot 160 dot 71. So, this is an example about it InetAddress related.

So, we have discussed about the URL, URL connection, HttpURLConnection and then InetAddress they were the concept basically regarding the actual specification of the distance server out of all even the server or any other machines that we are going to connect or establish a connection

(Refer Slide Time: 09:43)



```
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();
  }
}
```

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>
```

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket C
```

Now after having this information now it is our task to see exactly the as we have mentioned there are 2 protocols connection oriented and connectionless and they are basically implemented using UDP and then TCP IP; UDP for connection less and then TCP is for connection oriented these are example that we are going to discuss going to give you is that connection less protocol following the UDP. And here the basic idea is that we will prepare the 2 programs, 2 sockets rather using the DatagramSockets and then DatagramPackets layer.

And then the 2 sockets we will run concurrently in this at the same time in this case in the two sockets are in the same machine, but from the two different windows this means that assuming that this is basically they are running in a separate machines actually. So, in the first example as you see it is basically is the server sockets. So, it this is basically wait to receive any requests from one client and if it receives on request then it will basically responds.

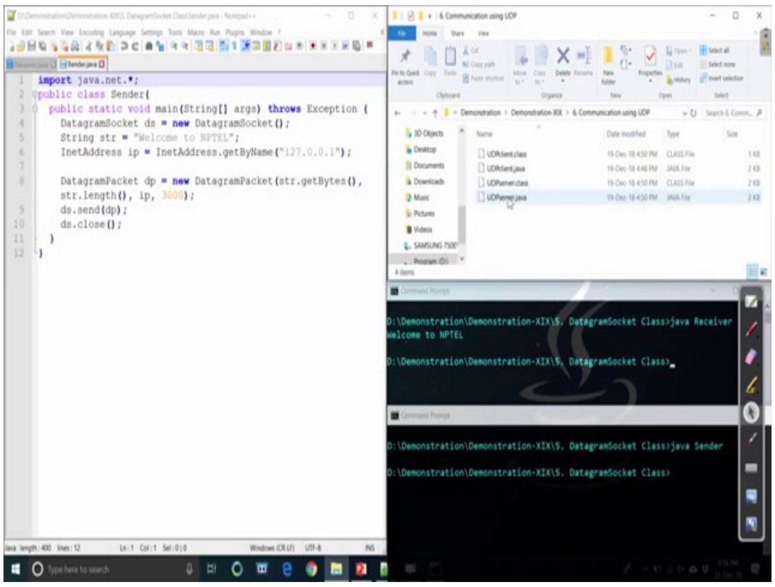
Here in this case as we see we create one object called the ds is a DatagramSocket objects and then you have to give the port number here you are giving that port number you can give any number here because it is implicit to give any port number. So, we have given one port number whatever it is there 4 digit, 5 digit whatever and by the by with our own today we have given 3000 (Refer Time: 11:25) so, no problem, but if this port number should be used for other communicating programs other sockets also.

Anyway, so, this is the basically buffer where you want to store we made that 1024 size of the buffer and we create a DatagramPacket here. As you know UDP follow DatagramPacket concepts that mean it will convert or enter the message into number of packets of peak size say 64 bytes like and then these basically packet we will create these things here is the 1024 is the total content the message and then its basically converting the packets actually.

And then here the this datagram packet is basically send they as a buffer like if we receive any message from the distance machine or from other sockets it will store into this packet. So, you receive the packet and then all the packet will be stored here in the dp and so, this is basically the receive is the method which is defined there in DatagramPacket concept itself. So, it will basically receive all the packets from the sockets and then that packet will be display here this is the method by which the entire packet content can be displayed on the screen.

We can save this packet into our local machine also using file output stream program and then finally, once the receive is over then we should close it and in this case server is waiting once it receive a packet it will receive it and then finally, then or sub so, it will close it. And so, this is the client side program a server side program.

(Refer Slide Time: 12:53)



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

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>java Receiver
Welcome to NPTEL

D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>
```

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>java Sender

D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>
```

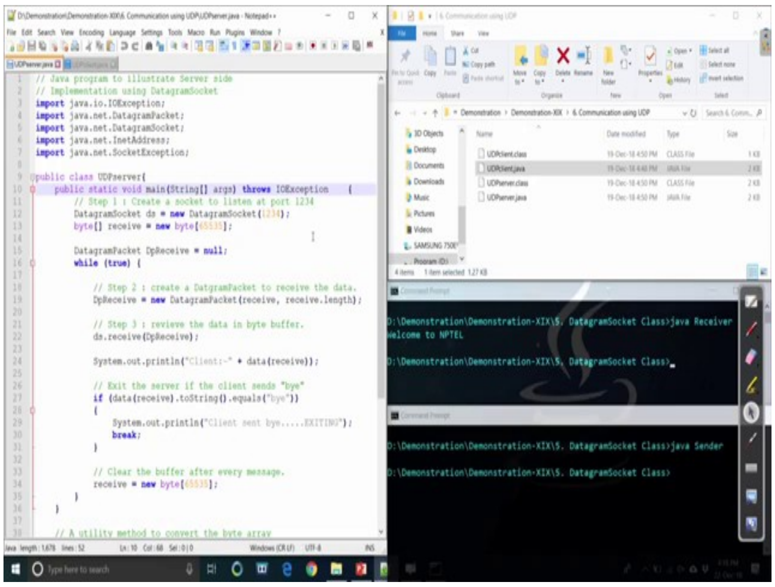
The client side program in sender whenever sender sends something how sender can sends it. Now the sender can sends it again DatagramSockets for the sender sides needs to be created. So, this is the ds object for the sender like and this is the String that message that the sender wants to send it smalls message ended here and then InetAddress is created ip because it is InetAddress is required and which machine as is the default one.

So, in the same machine all the InetAddress will be by default it is 127.0.0.1 and then DatagramPacket is basically for the message it basically converting the InetAddress, so, this is the object it is there. And so, this basically DatagramPacket is created taking this thing as a total information about the string it is here and the datagram packet is created with the address to whom this DatagramPacket should be send here is the address of the server we have already mentioned earlier and then datagram packet is ready to be sent.

And then here is the send method will send this DatagramPacket to the server basically to this port at port at this actually. And then finally, once send is over it will basically close the connection. So, this is a concept that sender and receiver. Here receiver being a server we will receive a message from the sender; sender is the client here in this case now here just I am going to give a demo of this program execution you will be able to understand how it will work here. Here the first we have to start the receiver program then you have to start the sender program means receiver should be there first; otherwise if you start sender program first then sender we will say sending this, but there is nobody to receive it, so, they are messaging will lost.

So, here as we see receiver, so, receiver receive this message welcome to NPTEL this is basically the windows which is running the receiver socket and this is the window which is running the server socket. So, in the server socket we just run the server or Sender actually and then Sender sends this document message as you know welcome to NPTEL like and it goes to the receiver, receiver received it and then print it. So, this is the idea about using DatagramPacket EDP protocol here datagram packet.

(Refer Slide Time: 15:11)



```
// Java program to illustrate Server side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class UDPserver{
    public static void main(String[] args) throws IOException    {
        // Step 1 : Create a socket to listen at port 1234
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while (true) {

            // Step 2 : create a DatgramPacket to receive the data.
            DpReceive = new DatagramPacket(receive, receive.length);

            // Step 3 : revieve the data in byte buffer.
            ds.receive(DpReceive);

            System.out.println("Client:-" + data(receive));

            // Exit the server if the client sends "bye"
            if (data(receive).toString().equals("bye"))
            {
                System.out.println("Client sent bye.....EXITING");
                break;
            }

            // Clear the buffer after every message.
            receive = new byte[65535];
        }
    }

    // A utility method to convert the byte array
```

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>java Receiver
Welcome to NPTEL

D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>
```

```
D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>java Sender

D:\Demonstration\Demonstration-XIX\5. DatagramSocket Class>
```

Now, our next example here this is the second version of this protocol here basically implementation of client server mechanism, but here the idea about is that. So, is a dialog message within this. So, it is a dialog message; that means, the same UDP protocol that we have discussed earlier, the difference here is that the communication we will just in a chatting manner; that means, first server socket program we will always be there to listen to the client. And, then client we will send some message server we will receive this message and then server again send some message to the client, client will receive it and this dialog may continue whatever the things until server or any client send a stop message for that.

So, this is the program consulate is here and you see this is the implementation of this one all these import section is mandatory java dot net and other things in input output for a streaming and everything it is there all those things is obvious. And this program as you see ds is basically the socket I am now discussing about server side program. So, ds is the socket for the server and here is basically port number is 123 port number is this one right.

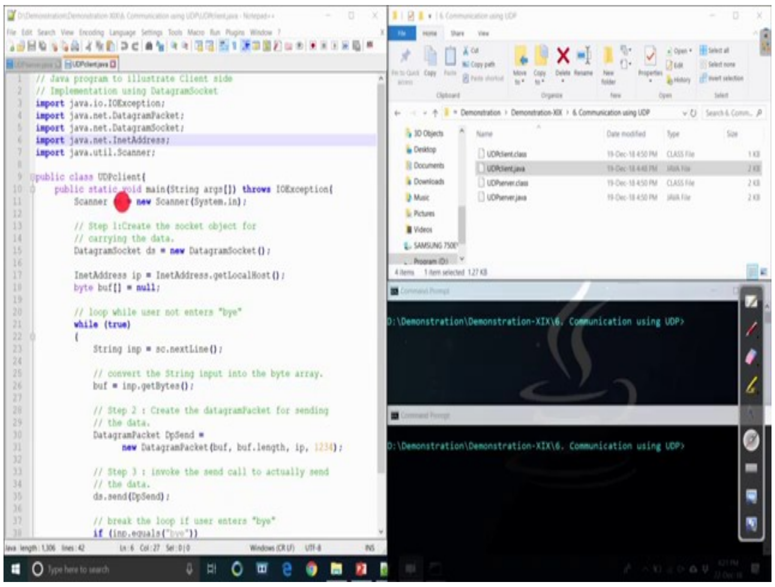
And then this is what is called the 123 is the port number right, here yeah this is the port number and this is the size of the maximum buffer that it can handle actually. So, it is a byte this is a basically temporary buffer we can say is an array of stream basically and then DatagramPacket DpReceived this is we say we create a DatagramPacket because we have to use the UDP.

So, datagram packet we have declared a datagram packet name the DpReceive and then DpReceive object is created that mean receive if we receive some message for receive is basically the content that needs to be received from a distance machine and the length of the packets actually and then ds dot receive is basically we have to receive if it receives any packet there.

So, this is basically the method which is basically in a constant watching and then whenever its receive a packet it will basically get this one. Once this packet is received and then you can send the prompt message that client and what are the message that is received is basically printed there; if data received to ok. So, if there is an message they by; that means, it will say that this is our tongue I mean communication is I mean over and then it will close the message.

So, this is the server side program, similarly there is a ok. So, let us go to the yeah fine. So, this is the client side program similar to this.

(Refer Slide Time: 18:13)



```
// Java program to illustrate Client side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class UDPclient{
    public static void main(String args[]) throws IOException{
        Scanner [illegible] new Scanner(System.in);

        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
        while (true)
        {
            String inp = sc.nextLine();

            // convert the String input into the byte array.
            buf = inp.getBytes();

            // Step 2 : Create the datagramPacket for sending
            // the data.
            DatagramPacket DpSend =
                  new DatagramPacket(buf, buf.length, ip, 1234);

            // Step 3 : invoke the send call to actually send
            // the data.
            ds.send(DpSend);

            // break the loop if user enters "bye"
            if (inp.equals("bye"))
```

```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>
```

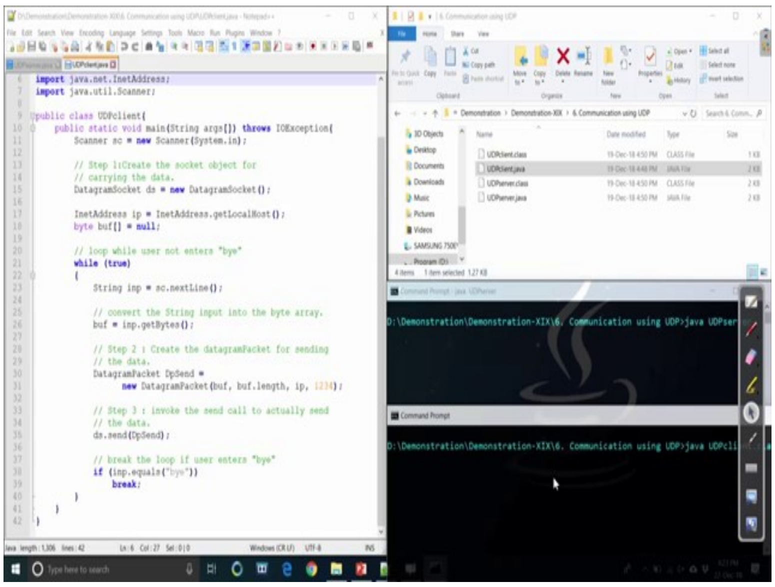
```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>
```

In this case as you see here I want another socket client socket is created and this client socket is basically read the continuous steaming from the keyboard like we whatever it is type it from the keyboard it will read it this is the Scanner class is created here. The

Scanner class is basically reading some input from the keyboard standard input and here is the socket that we have created datagram sockets and then this is the ip address of this thing. And this is the temporary buffer where the before sending this message you will needs to be stored here and this is the basically portion of the communication.

So, this is the communication here and is basically. So, is the reading wants input from the keyboard. So, this is the Scanner only and then its store in the buffer and then it basically convert in the packet. So, this is the DpSend is the packet where the buffer the packet will be created buffer will content all the message that the client wants to send it to server.

(Refer Slide Time: 19:19)



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

public class UDPclient{
    public static void main(String args[]) throws IOException{
        Scanner sc = new Scanner(System.in);

        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
        while (true)
        {
            String inp = sc.nextLine();

            // convert the String input into the byte array.
            buf = inp.getBytes();

            // Step 2 : Create the datagramPacket for sending
            // the data.
            DatagramPacket DpSend =
                  new DatagramPacket(buf, buf.length, ip, 1234);

            // Step 3 : invoke the send call to actually send
            // the data.
            ds.send(DpSend);

            // break the loop if user enters "bye"
            if (inp.equals("bye"))
                break;
        }
    }
}
```

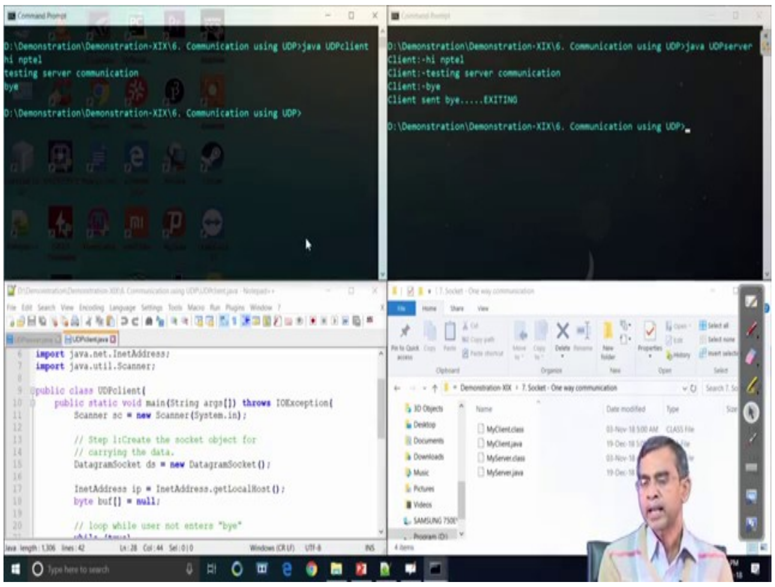
```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>java UDPser
```

```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>java UDPcli
```

And then finally, ds send the socket we will send the packet to the server and then it will basically continue until this message by is there. So, if it types by then both server will understand that client is no more interested to send any message and then the server also will close its connection as well as this client also will close its connection.

So, it is basically the idea about these are server and client side program, this is the client side program. Now here we are just see again two windows we have to first start the server window fast. So, the server is ready to listen any request from the client and then we will start the client server client program and another window will be there. So, two windows actually we can write one here, and one here.

(Refer Slide Time: 20:21)



```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>java UDPclient
hi nptel
testing server communication
bye

D:\Demonstration\Demonstration-XIX\6. Communication using UDP>
```

```
D:\Demonstration\Demonstration-XIX\6. Communication using UDP>java UDPserver
Client:-hi nptel
Client:-testing server communication
Client:-bye
Client sent bye.....EXITING

D:\Demonstration\Demonstration-XIX\6. Communication using UDP>
```

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

public class UDPclient{
    public static void main(String args[]) throws IOException{
        Scanner sc = new Scanner(System.in);

        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
```

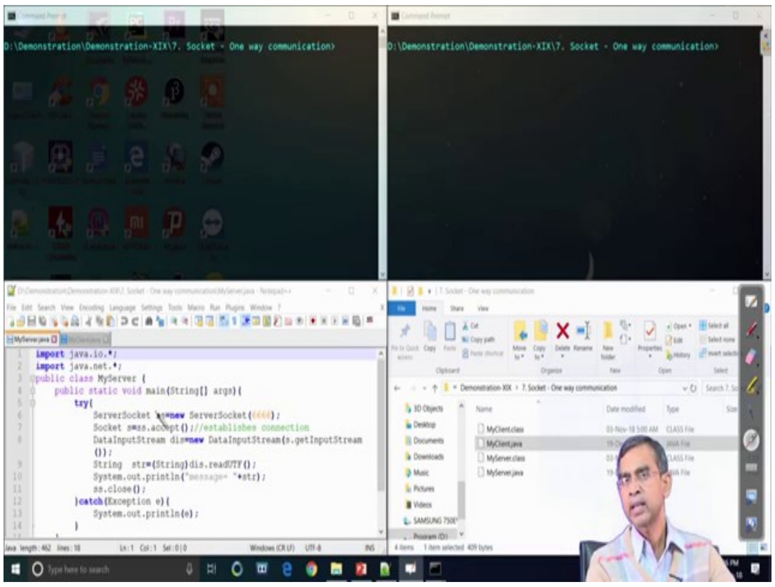
Now, so, this is basically the client side program. So, this window is basically client window and this is the server window, now here clients would send some message. So, hi I just are typing the message hi nptel java program client from client.

Now I am sending this one yeah now the time message that whatever it is there we are typing hi nptel yeah I am from client ok. Now you see we have typed hi nptel I am from client then ok, server should send some right then it can send some more message like nptel test write something testing server ok. So, it is there fine yeah. So, whatever the message that we are typing we are sending it now if we type by then ok.

So, here in the client send by and exiting this is the one right way by which the server a client can send some message in a continuous steaming manner it is not the dialogue right ok. So, it is not the dialogue of course, the dialogue program will come there. So, the both way the communication can be shown their using some TCP protocol we have planned it anyway.

So, this is basically the message that, so, the machine can be done here. Now, let us come to the second example our next example is basically to explain the TCPIP protocol using again server sockets and then socket that we have discussed.

(Refer Slide Time: 22:39)



```
D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>
```

```
D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>
```

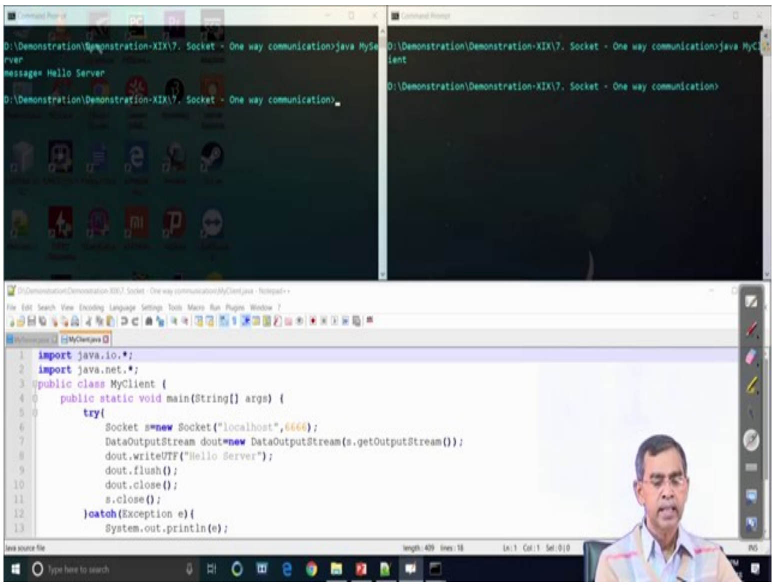
```
import java.io.*;
import java.net.*;
public class MyServer {
    public static void main(String[] args){
        try{
            ServerSocket ss=new ServerSocket(6666);
            Socket s=ss.accept();//establishes connection
            DataInputStream dis=new DataInputStream(s.getInputStream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

So, server socket is for the server side programming and the socket classes for the client side programming. Now as you see here this one ok. So, here as you create a ss is basically the server side sockets and this is basically we have given on port number to the socket program as 6666. So, it is given by us and this port number should be used by the client if this client wants to communicate to the server again. And this basically is the socket object is created; that means, a socket basically s one socket object is created only if the server socket accepts the connection that if it is reaches to him.

So, here basically ss.accept is basically the request that can be attended by the server and accepted. So, on socket object is created regarding to that one. So, this is basically server socket is now ready and then socket also from the client is also accepted by the server it is also ready. Once everything is ready yeah, so, once everything is ready then we have to read the message that will be send by that will be send by client. So, it is basically we have to create a data input stream dis and you see s means if we receive something from the ss not get input stream.

So, get interesting method is basically one method it is there in the socket class itself and this method we will read all the content that is there from the socket program sites actually and then this will basically read as a string has we have given this one. So, it will store as a string whatever the input stream that we can read converting the string actually and then finally, this string will be printed on the screen and then the server socket will be we will closed. So, server socket if we run this program it will wait to receive on request and it will accept the request and then you read the content from that socket and then finally, print it.

(Refer Slide Time: 24:51)



```
D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>java MySe
rver
message= Hello Server

D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>_
```

```
D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>java MyCl
ient

D:\Demonstration\Demonstration-XIX\7. Socket - One way communication>
```

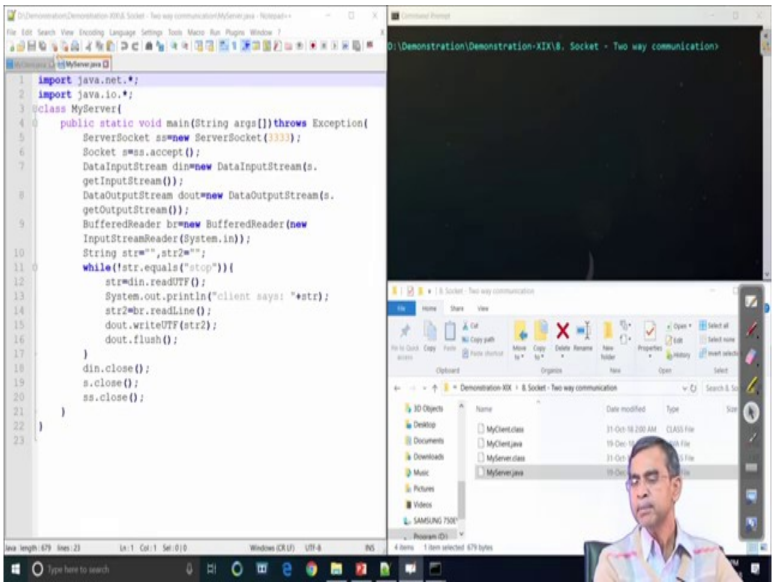
```java
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) {
        try{
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
  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

So, this is about the server program in this case and as you have seen that we have use the server socket and the socket. So, it basically follows the connection oriented protocol namely TCP protocol and here is the client side program. So, for the client side program is concerned here as you see we create a server a Socket for the client. So, this is and the port number is the same as the previous one and then this is the output stream that you have created because you have to send something.

So, gateOutputStream method is basically we will create a stream with the string that is there. So, here d out right UTP Hello Server this is the string actually the message will be converted into output form and then output stream will be maintained. So, that the steam will propagate I will send to the server and finally, it will close. So, server is closed that is all this is a message here is basically very similar to the UDP protocol where server sends a message to the client and then client printed it.

Whether it is a continuous manner or is it just only one chat whatever it is there. So, in this case also it is write one chat is their server and client both are ready in to windows and then client send some message the message is here Hello Server and the message is received by the server and then printed it that is all.

(Refer Slide Time: 26:21)



```
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 str="",str2="";
        while(!str.equals("stop")){
            str=din.readUTF();
            System.out.println("client says: "+str);
            str2=br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }
        din.close();
        s.close();
        ss.close();
    }
}
```

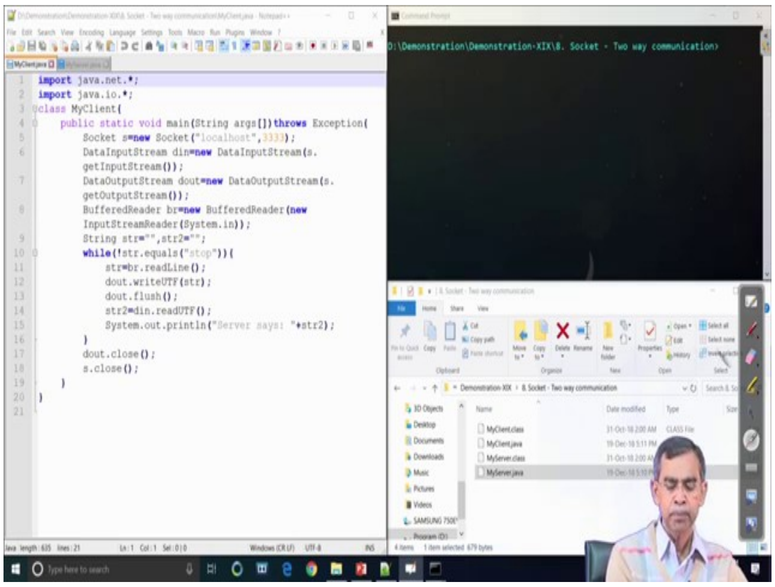
```
D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>
```

Now, the next example is basically dialogue, this is more interesting program on and here the dialogue between the 2 sockets; one is ServerSocket another is a client socket and then client will send some message server will received server will print it then it will sent to the client, client will print it then it will send another message, the communication may continue until both the party may want for this. So, there will be some stopping criteria say stop from stop message from both sides then both party will understand that chat is over.

So, here is the example very similar to the previous one only absolute there is no difference as you see here this is the ServerSocket the port number of the server program and this is basically accept to the sockets right so; that means, if it accepts then a socket object will be a established a connection from the server to that a Sockets one.

And then this is basically Input; that means, reading stream from the distance machine I mean from the other Sockets and then Outputting; that means, if the server wants to send something then it basically outputting. And here is basically flow that it will display the content and this content will continue until there is a stopping criteria reach there is a stop; that means, if it miss receive a message stop then this will basically close and then this is the server side one.

(Refer Slide Time: 27:41)



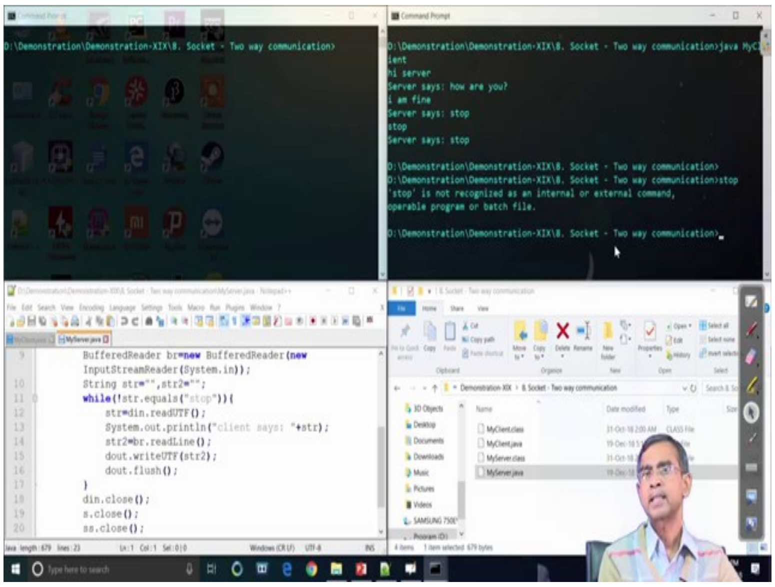
```
import java.net.*;
import java.io.*;
class MyClient{
    public static void main(String args[])throws Exception{
        Socket s=new Socket("localhost",3333);
        DataInputStream din=new DataInputStream(s.
        getInputStream());
        DataOutputStream dout=new DataOutputStream(s.
        getOutputStream());
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        String str="",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();
    }
}
```

```
D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>
```

Similarly, there is a client side program as you see here the client side program these are the socket that we have created the post port number this one and this is basically in the same machine. So, we have to tell that this is the localhost. If we say explicitly specify the ip address or some host name then; obviously, it will go to that distance that is the only difference. So, in this case the localhost means in the same machine, but from the 2 windows that is how you have used it and then rest of the things is very similar to there we have to define the data input stream; that means, the reading from the server side any message and then output is basically sending some message to the server using TCP protocol.

And here is that basically if the client received any message then how they will print it how they process it. So, this is the continuous manner means both the party will continue their dialogue till there is a stop from both sides or from anyone sides like. So, this is the example now we can give an example. So, that we can see we will type some message from the client it will go to the server, server we will receive it server we will send some message.

(Refer Slide Time: 28:47)



```
D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>
```

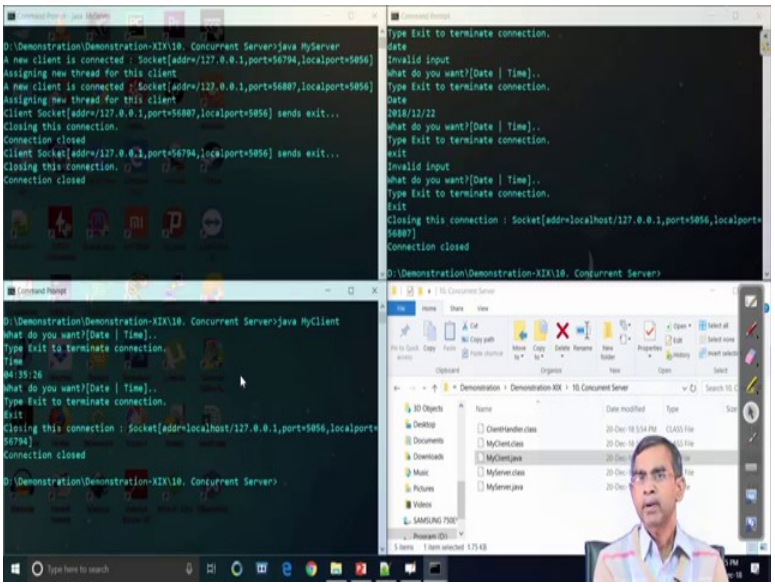
```
D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>java MyCl
ient
hi server
Server says: how are you?
i am fine
Server says: stop
stop
Server says: stop

D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>
D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>stop
'stop' is not recognized as an internal or external command,
operable program or batch file.

D:\Demonstration\Demonstration-XIX\8. Socket - Two way communication>_
```

```java
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        String str="",str2="";
        while(!str.equals("stop")){
            str=din.readUTF();
            System.out.println("client says: "+str);
            str2=br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }
        din.close();
        s.close();
        ss.close();
```

It will come to the client, client will again receive it and then client will type something like this. So, now, here from the server side ok. So, server and client both are now ready now here from the; so, we are just sending from the client side hello server, hi server end. So, here you see client says hi server, now either from the client or server we can write something now in the client hi how are you; how are you I am sending this message now is go to I am fine ok. So, its come to this I am fine and then if we come here in the client or server then buy or stop.

So, now we will see this will go there. So, client server say stop and then yeah. So, anyway so, this top basically message received from either server or client or from both basically close the communication it is there. So, this is the idea about the dialogue message. Now so, and then remote communication is basically the same thing only the thing is that in so, localhost we have to mention the ip address explicitly.

(Refer Slide Time: 30:05)



```
D:\Demonstration\Demonstration-XIX\10. Concurrent Server>java MyServer
A new client is connected : Socket[addr=/127.0.0.1,port=56794,localport=5056]
Assigning new thread for this client
A new client is connected : Socket[addr=/127.0.0.1,port=56807,localport=5056]
Assigning new thread for this client
Client Socket[addr=/127.0.0.1,port=56807,localport=5056] sends exit...
Closing this connection.
Connection closed
Client Socket[addr=/127.0.0.1,port=56794,localport=5056] sends exit...
Closing this connection.
Connection closed
```

```
Type Exit to terminate connection.
date
Invalid input
What do you want?[Date | Time]..
Type Exit to terminate connection.
Date
2018/12/22
What do you want?[Date | Time]..
Type Exit to terminate connection.
exit
Invalid input
What do you want?[Date | Time]..
Type Exit to terminate connection.
Exit
Closing this connection : Socket[addr=localhost/127.0.0.1,port=5056,localport=
56807]
Connection closed

D:\Demonstration\Demonstration-XIX\10. Concurrent Server>
```

```
D:\Demonstration\Demonstration-XIX\10. Concurrent Server>java MyClient
What do you want?[Date | Time]..
Type Exit to terminate connection.
Time
04:35:26
What do you want?[Date | Time]..
Type Exit to terminate connection.
Exit
Closing this connection : Socket[addr=localhost/127.0.0.1,port=5056,localport=
56794]
Connection closed

D:\Demonstration\Demonstration-XIX\10. Concurrent Server>
```

Now, I will quickly come to the concurrent what is called the server let us go to the concurrent server. And, here is basically here as the distance meeting is not possible here to give a demo we will give basically in the same machine I am giving I detail program you can have it from my sites you can get the program and you can run it I do not want to discuss about the detail program I will only give the demo of this execution.

So, here the idea is that one server program is there I can say the concurrent server it is basically thread implementation of the server which is be difficult which I have discussed in our theory class. And then here we can plan what are 2 clients that more than two clients also can be possible 2 clients for this 2 windows actually we have to do it. So, there are 2 windows that we have mention here. So, they are total altogether three windows one windows for the server and another windows for another two windows is for the client here now as you see here ok.

So, as you see here. So, this is basically the server window and this is the client 1 window and this is the client 2 window. Now so, all programs are running so, server is concurrent here in that because if the message comes for any one client it will response to that one. So now, I am typing something from the date about yeah, so, client is now communicating in ok.

So, here basically so, client basically give the date, so, date has been printed then it basically ask that what do you want then whatever it is there right time I can say right time. Now here basically the idea is that client 2 basically asking the time now all these response is coming from the server itself right. So, server is replying that it asks the date, this is ask the time and they are receiving this one and this is a message actually it is that ok.

So, here what I can say is that client is basically responding to any request from any clients and then is basically the input is input to the server is basically the request from the client regarding any information and then it receives and then printed that is all. So, this way we can see that these are concurrent program is there and details of the program that you can have it from the program exact details from the slide discussion as well as from my sites itself. We can download it and then you can run and you can try to understand how it works and you should its practice it then you can understand much more about it ok.

Thank you very much.