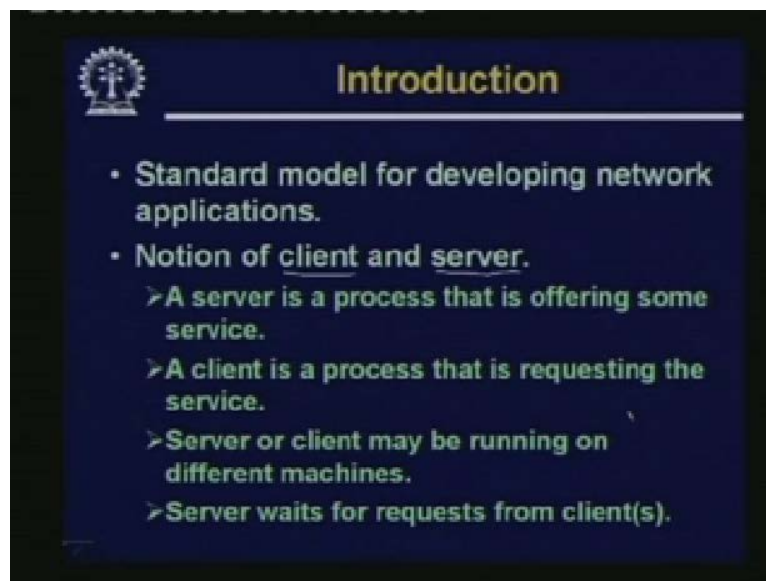


Internet Technology
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No #30
Client-Server Programming in Java (59:52)

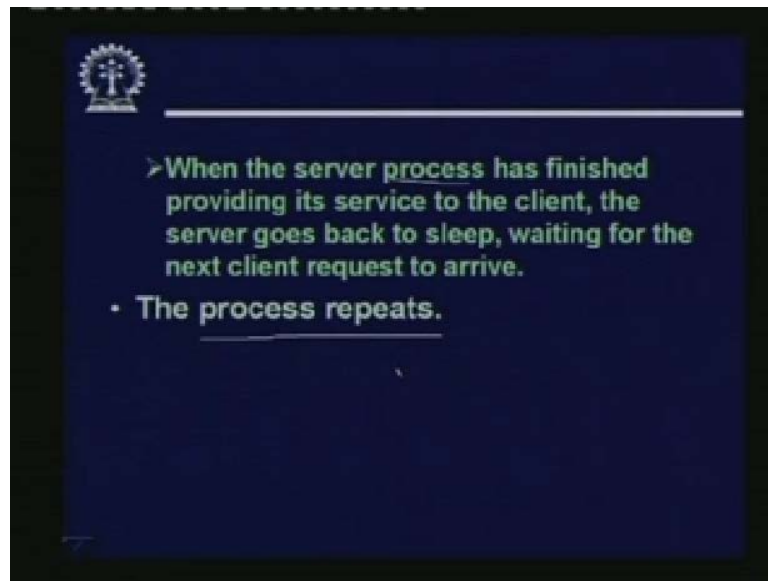
In this lecture we shall be talking about client-server programming in Java. Now if you recall from our last two lectures we had looked at the various features of Java. We shall we had seen that how some of the features of Java makes it suitable to be used in a platform independent way as a language which can be used very conveniently over the internet. Today we shall see basically how to create a or how to write network programs in Java where by two Java programs over a network can communicate among themselves. Now you recall earlier we had talked about some of the concepts of client-side programming. So before we actually go in to how we can write such network programs in Java. Let us have a quick recapitulation of our basic concepts on client-side programming. So let us have a quick recap of the client-side model.

(Refer Slide Time: 01:56)



The client-side model as you know is a standard model which has been accepted by many for developing network applications. Now in this model there is a notion of a client and there is the notion of a server. As the name implies a server is a process or you can also regard it as a computer in which the process is running. So a server is such a process which is trying to provide services to other entities which are called clients. So a client will be a process which may be running on the same computer or some other computer which is requesting some service. As I said in general the client and server may run on different machines. So a server will be waiting for request from clients. If a request comes the request will be serviced and the server will again go back to serve the next client request.

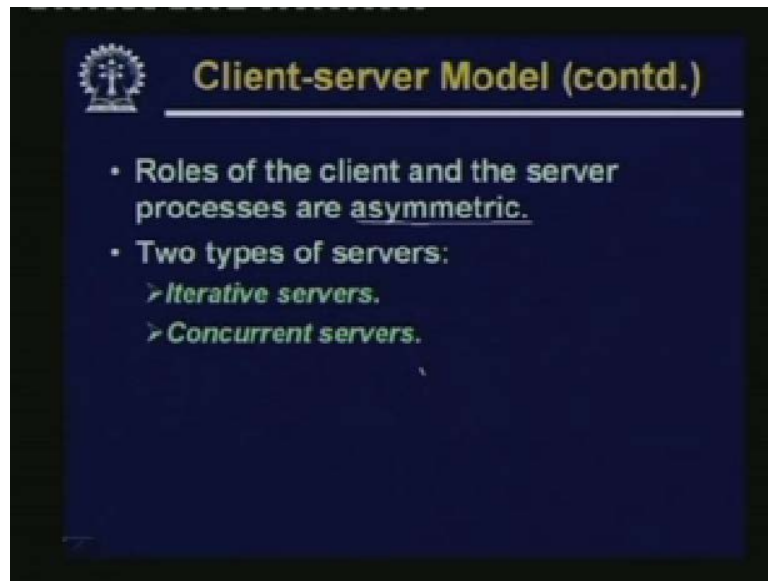
(Refer Slide Time: 03:06)



Pictorially the model looks like this you have a server out here and there can be several clients which may be sending request to the server. So again the way you design a client-side programming environment it depends on the kind of service the server is trying to provide to the clients whether the service is definitive in the sense, that you can predict how much time will take to serve a request or the request can come in any random order and times will all be. You can say indeterminate you cannot predict beforehand. So depending on number of issues the exactly the way, how the server should respond to a client request will differ we shall see this later. So in the client-side model, if you look at a typical scenario, the server process would start on some computer system. In fact the server should be executed before the client.

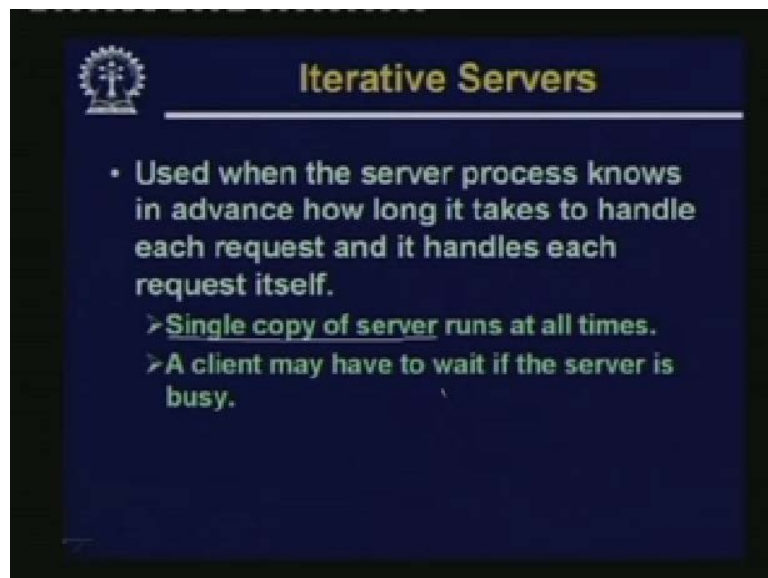
Server usually makes some initialization and then goes to a wait state or a sleep state where it waits for a client request. Now after that a client process can start on some machine and whenever it wants some service from the server it will send a request to the server. The server will receive the request will process it and after it has finished processing or providing the service. The server will again go back to sleep waiting for the next client request to arrive. And this process will repeat as long as the server process is running. So server process is one which is continuously executing on a machine and it is mostly in a waiting state waiting for a request for a request for the client and whenever such a request comes the server can immediately serve the request. And then again go back to the waiting state for the next request to arrive.

(Refer Slide Time: 05:31)



So as I had said, depending on the environment you can have different types of servers. Now in general if you look at a client program and a server program you will find that there are some differences in a way you write them. So in that sense the client and the server program or processes are asymmetric in terms of their rolls or in terms how you write. And depending on the environment and the kind of service you can categorize the servers in to either Iterative servers or Concurrent servers. Let us see the basic idea behind these two.

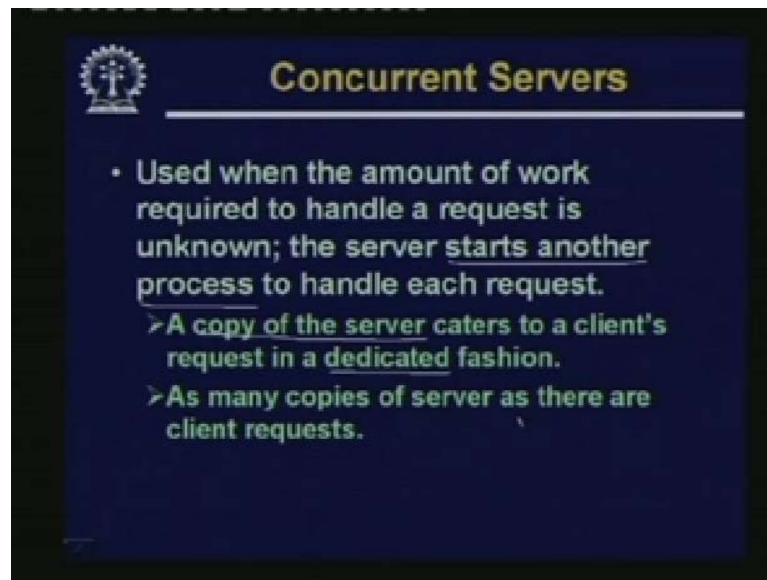
(Refer Slide Time: 06:17)



Iterative servers are used when the server process knows beforehand in advance, how long it will take to provide a service. And the server process directly can handle each of the requests. The idea is that suppose I am the server and if I know exactly how much time I would take to provide a service and of course it is also true that this total time is not very large. This is within a reasonable limit, then what I can do as a server that without asking anyone else I myself can directly serve the request and then again go back to serve the next request. This

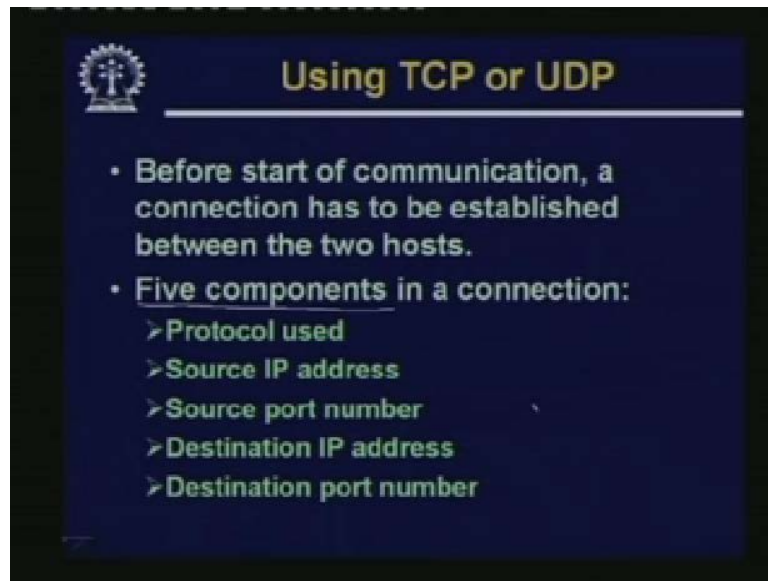
would be acceptable because I know beforehand how long I would be occupied with processing this particular request. And secondly, since the time as I had said is reasonably less the next client, who also had put up a request will not have to wait for too long time. So in this kind of a server characteristics are these is a single copy of the server which runs on the machine at all times. And the clients will be sending requests and this single copy of the server will be providing service in an iterating fashion. So if the server is busy at some point in time a client sending a request will have to wait until the request has been serviced.

(Refer Slide Time: 07:58)



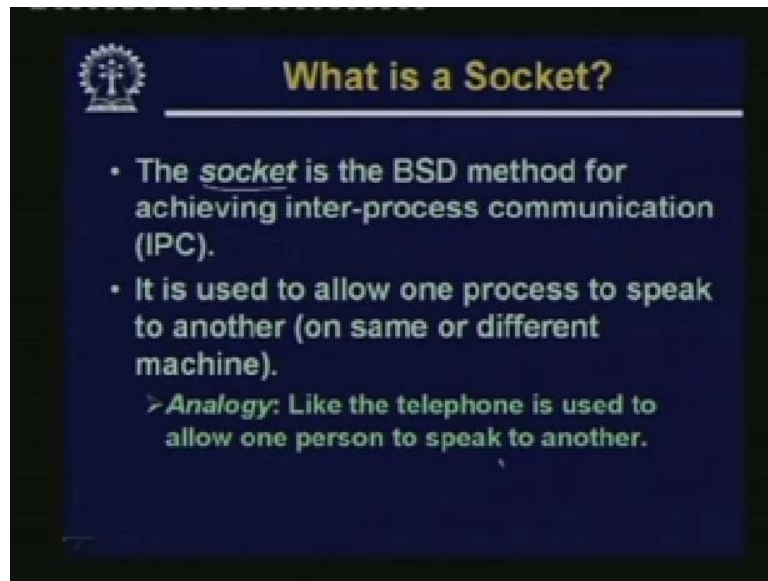
In contrast a Concurrent server is more general. This kind of a server you can use when the service time is unpredictable and maybe large. So you can this when the amount of work required to handle a request is not known or cannot be predicted beforehand. So here the way the server works is that whenever a request comes. Server will start another process dedicated to that request. So the idea is like this, suppose I am a server, now if a request comes I shall create another process and I shall ask that process to handle that client request in a dedicated fashion. And myself I will again wait for the next request go back and wait. So if the next client request comes again. Now I will again create another process and ask that process to handle the second client request dedicated way. So there are 10 concurrent client requests. I can create 10 concurrent processes to handle all of them together the server can provide service to a number of clients at the same time simultaneously. So I had said several copies of the servers are made and a particular copy will cater to a particular client's request in a dedicated fashion. There is no question of sharing. So if there are n numbers of client requests. There will be n copies of the server program that will be created.

(Refer Slide Time: 09:42)



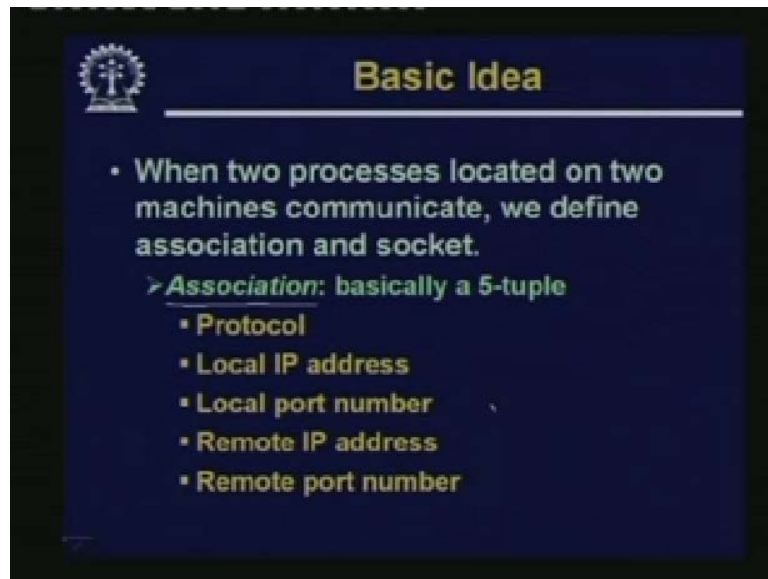
Now another choice you need to make here that whether you want to use TCP or UDP for your communication. Now you know the basic you can say differences between TCP and UDP when you should use them. If you want that the protocol itself should handle reliability flow control. Guaranteed order of delivery of the packets then you should use TCP. But if you have some kind of application where even if some packets are lost we do not care or you want that all these services which TCP provides normally you want to provide them in your application. There then possibly you can use UDP because in general UDP would be faster. Because UDP will always try to find out path through the network and will try to deliver packets as fast as the network can. So you recall we had said earlier that whenever we establish a connection either using TCP or UDP. There are five components that need to be specified before we can say that a connection has been established. The Protocol TCP UDP or any other proprietary protocol you may be using you have to specify that source IP address and the port numbers. This will uniquely identify the source process. Similarly destination IP address and the destination port number. These five components, if you take together they will constitute a connection.

(Refer Slide Time: 11:26)



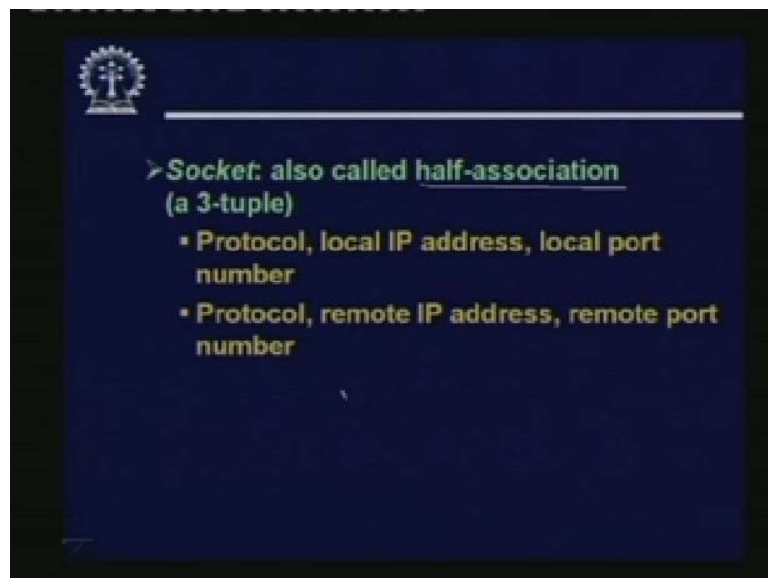
Now from the concept of connection, the concept of socket emerges actually the socket is a term which was initially coined by the Berkeley Software Distribution which actually provided a set of functionalities for inter-process communication on the UNIX system. A socket basically is used to allow one process to speak to another. The processes may reside on the same link or on different links of different machines. Now as an Analogy, now opening of a socket is like making a telephone connection. So once you have a telephone connection, you can simply speak on your handset and the person on the other side will be able to listen. Now socket using a socket is convenient because once you have opened a socket by specifying all the relevant information, now we can treat the network connection exactly similar to the case as you are accessing your file, like you are reading or writing from a file. In the same way you can read from a socket or write in to a socket. The socket the way you have created them, that details will be used by the system to automatically handle the network connections that is there underlying. So socket is some kind of an abstraction you can use at the transport layer level. So when you are writing your applications or developing applications in makes your task simpler.

(Refer Slide Time: 13:09)



Now this as I had said that when you say a connection is there. The five things that we had just mentioned which constitute a connection in the Berkley Software Distribution terminology, this is called an Association. This five tuple is called an Association. Now with respect to a network connection if you look at the 2 sides of the connections separately.

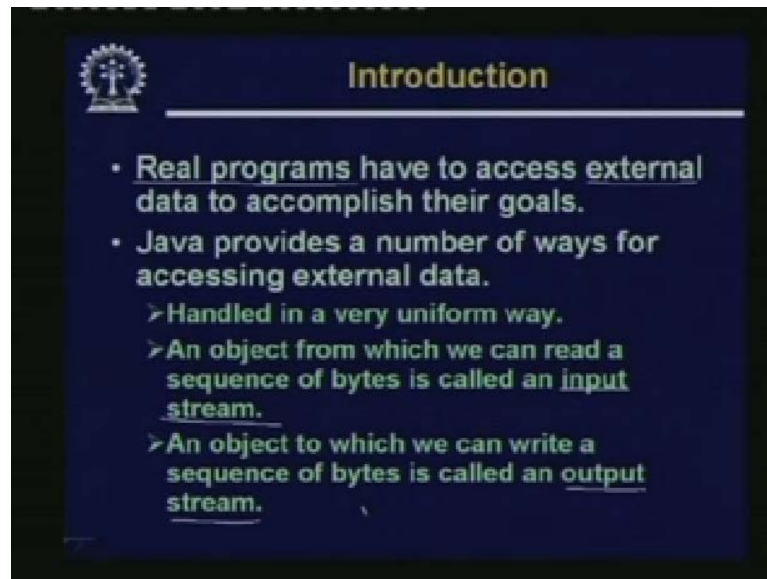
(Refer Slide Time: 13:39)



Then the concept of socket comes in which is also sometimes called half-association. A socket basically consists of the protocol type and the IP address and port number of any one side of the connection either local or remote. So there will two sockets on the two sides, both taken together will be defining the connection. So not let us look us look at with this background that how we can create network programs in Java. What are the features and what are the functionalities that the language Java provides for doing such a thing. Now let me tell you here that you will see that in Java the way we write network programs. It is very similar to the way we access files. This is also true for any other programming language like

C, C++, whatever you think of. But in Java these similarities much more pronounced in terms of the objects the methods you use for the purpose, let us see.

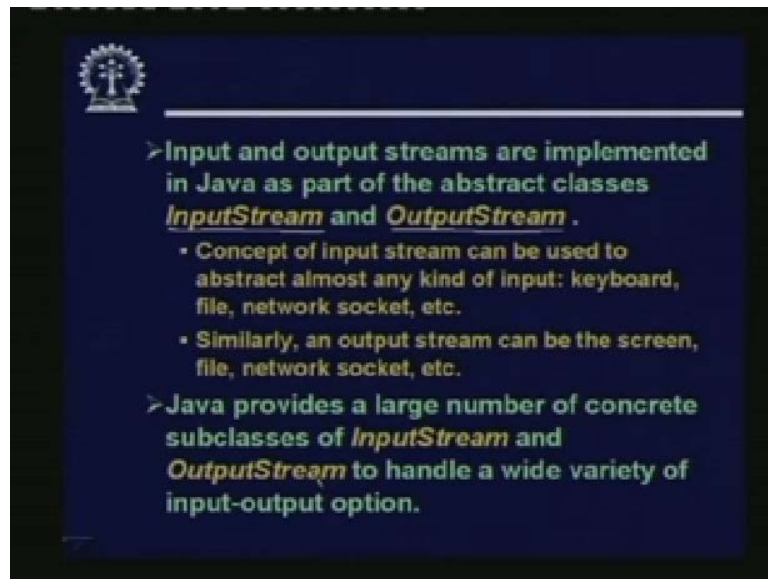
(Refer Slide Time: 14:49)



So the basic background says that means when you develop some real world applications. Such real world applications will not only rely on data that are available locally on your computer. So they will typically have to access external data to provide some meaningful service to the users. Java as a language provides a number of libraries and classes to facilitate accessing external data in a number of different ways. Now there is one common feature here. Accessing of external data in Java is handled in a very uniform way. Uniform way means it does not make any distinction whether you are for from a file reading from a keyboard reading from a network socket.

So the way you basically handle such input or output operations in uniform across all types or all sources of such input or output data. So in Java terminology there is something called a input stream and an output stream we define input stream and output stream. Now the way you define is as follows. An object from which you can read a sequence of bytes is called an input stream. As I had said this can be a file this can be a keyboard. This can be a scanner; this can be also a network socket. Similarly an object where you can write a sequence of bytes we call it an output string. So when you develop any kind of network or high end applications in Java. You are actually concerned about the input streams and output streams how to handle them.

(Refer Slide Time: 17:00)



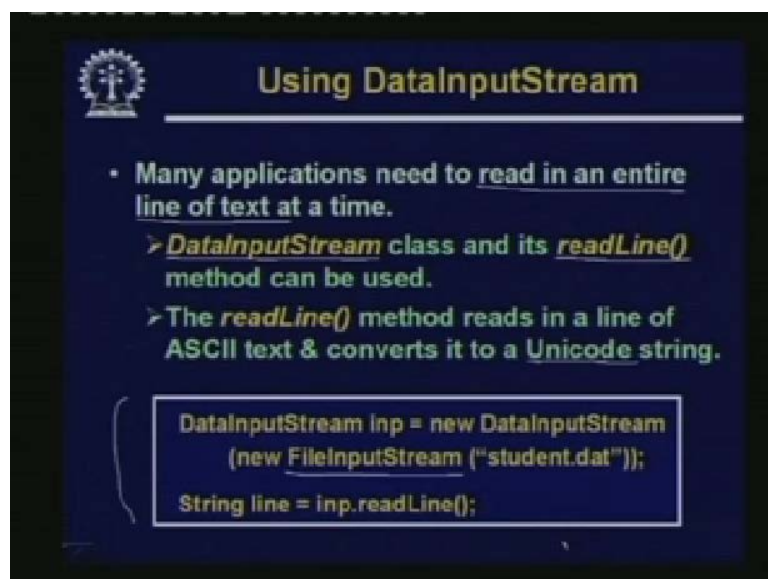
> Input and output streams are implemented in Java as part of the abstract classes *InputStream* and *OutputStream*.

- Concept of input stream can be used to abstract almost any kind of input: keyboard, file, network socket, etc.
- Similarly, an output stream can be the screen, file, network socket, etc.

> Java provides a large number of concrete subclasses of *InputStream* and *OutputStream* to handle a wide variety of input-output option.

So in Java such input and output streams are implemented using the abstract classes which available the names of these classes or input stream and output stream. As I had said the concept of the input stream, we can use to abstract almost any kind of input sources keyboard file network socket or any other. Similarly an output stream can be used to abstract almost any kind of output devices which can be the screen, which can be a file, it can be a network socket it can be plotter anything. Java as a language is very rich in providing such input or output capabilities in the sense that I provide a large number of concrete subclasses of input stream and output stream. See input stream and output stream are generic classes for inputs and outputs of bytes. Now under input stream there can be a separate subclass for networks separate subclass for files, a separate subclass for console I/O and so on. So Java provides you with all these facilities of subclasses using which you can write or develop the application in whatever way you want.

(Refer Slide Time: 18:25)



Using *DataInputStream*

- Many applications need to read in an entire line of text at a time.
 - > *DataInputStream* class and its *readLine()* method can be used.
 - > The *readLine()* method reads in a line of ASCII text & converts it to a Unicode string.

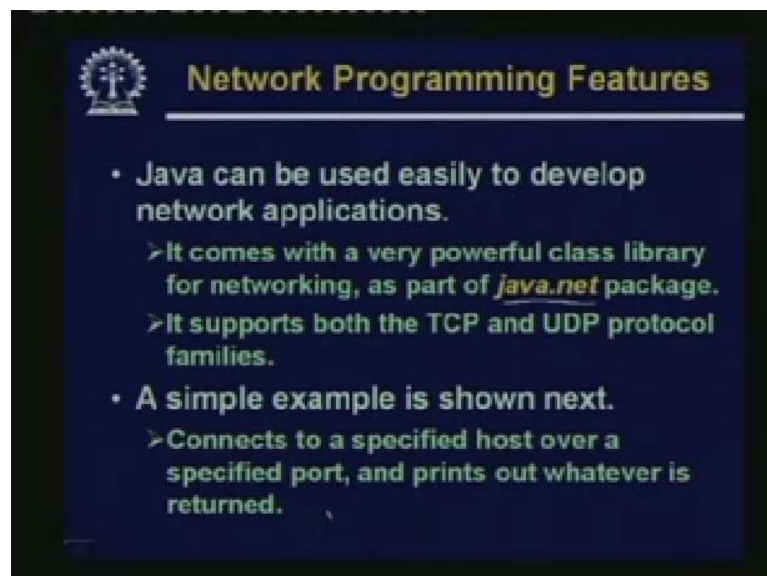
```
DataInputStream inp = new DataInputStream  
(new FileInputStream ("student.dat"));  
String line = inp.readLine();
```

So now let us see how in Java we can actually write or utilize these facilities. First let us see the concept of input stream. How we can use the data input stream concept in Java? Now here the example that you site is one where an application is trying to read in an entire line of text. Now when you want to do so, you will have to use a class called `DataInputStream` and a method which is there as part of the class called `readLine`; `readLine` is used to read one line of text from some input source at a time. When you say one line it means it will read the characters up to the end of line character. But when it is returned back it depends on actually the kind or way you store it whether you actually store the end line character or not.

So just as I had said this method will be reading in a line of ASCII text and store it and as I mentioned earlier in Java. The characters are stored as 16 bit Unicode characters. So it will be a Unicode string. The example shown here, actually illustrates how you can read a line of text from a given file. Here you are creating a variable input of type `DataInputStream`. This is new object you are creating from the class `DataInputStream`. New `DataInputStream`, the kind of `DataInputStream` you are specifying here, this is the subclass of input stream file input stream. File input stream takes a parameter which is the name of the file. File input stream implicitly assumes that the file you are opening, that will be opened for input.

So actually when the file is opened you cannot actually start reading the characters from them. So the file you are opening you can either use the file input stream or the file output stream depending on which one you are using the file will be either opened as input or output. So here we are creating the subclass or an object file input stream and after doing that this inp will be the handle or the object that will be returned. You can call `readLine` on this object inp, which will be returning a line of characters of type string. So as you can see it is fairly simple to read a line of character from any source. Only you will have to know that, what is the name of the subclass you need to use for handling that particular kind of input source.

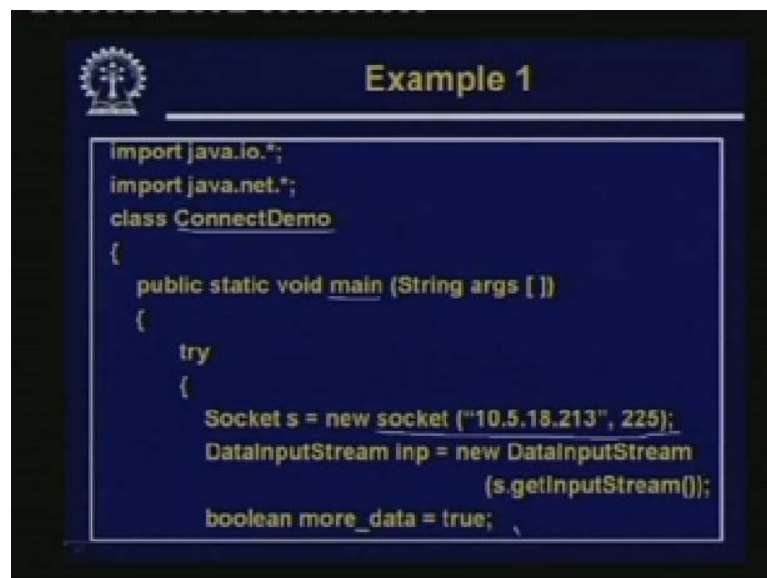
(Refer Slide Time: 21:25)



Now let us see that what are the network programming specific features that are available in Java. As I had before that the language Java has some facilities which are specifically meant for developing network applications. It comes with a very powerful class library for networking. That is there as part of java dot net package which you have to include in a

program if you want to have a network input output methods being used. So java dot net is the package you have to include at the beginning of program. So we will show a simple example how we can write a Java program which can communicate over the network. Now the example, will show next is like this. This particular Java program connects to a particular host over a specified port which means we are actually trying to write a client kind of a program. And what the client will do after connecting whatever the server will be returning it, will simply be printing it out on the screen. So actually we shall be illustrating how to write a client program which will be contacting a server. Now one thing you remember, whenever a client wants to connect to a server, there are two things you need to specify. The IP address of the server and port number on which the particular process on the server need to be contacted. So this client will be using these two information to establish a connection and after the connection is established. The server will be responding back with some string and whatever string comes back will displayed on the screen. This is what this example will show you.

(Refer Slide Time: 23:26)



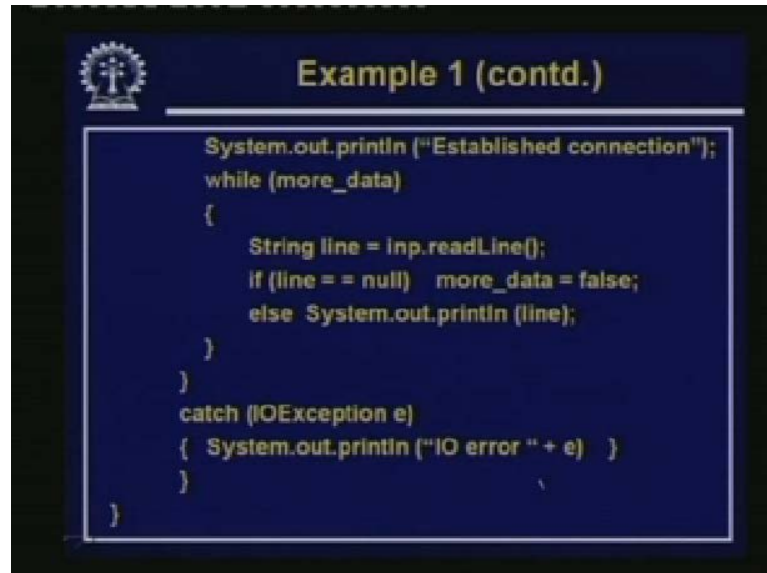
```
import java.io.*;
import java.net.*;
class ConnectDemo
{
    public static void main (String args [ ])
    {
        try
        {
            Socket s = new socket ("10.5.18.213", 225);
            DataInputStream inp = new DataInputStream
                (s.getInputStream());
            boolean more_data = true;
        }
    }
}
```

So the example starts like this. You see since this contains some input stream or output stream. So I have the java dot io package is included also, since we using some networking functions or method java net is also there. So here we define a class ConnectDemo, this is a Java application we are illustrating not a applet. This is the main method, in main method in the try block we are opening a so called socket using the socket class. Socket is a build in class which if you want to open you will have to specify two things; the IP address and the port number. Now the socket class is to be used by the client program you should remember this because when the client program starts it should start by trying to connect to the server where you will be specifying these two things.

This IP address and this port number so as an example I taken the IP address as 10.5.18.213 and the port number as 225. So you create a new socket and you return it as s, this s will be the new Socket object. Now you can use the standard data input stream class. This we had illustrated for the file example see here it looks very much the same only difference is I am using s here that means Socket. Socket I am opening by calling the getinputstream method write. So this inp is defined to be an object which actually represents an input stream object.

But the input is taken from the socket you have defined and since we will be displaying the data coming back. So I am defining a Boolean variable more_data which indicates that whether there is more data to be displayed or not. So this is initially set to true.

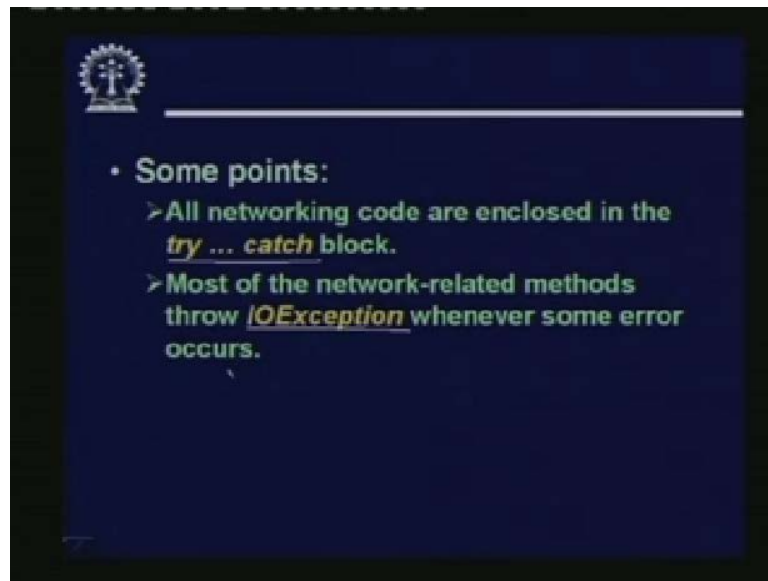
(Refer Slide Time: 25:53)



```
System.out.println ("Established connection");
while (more_data)
{
    String line = inp.readLine();
    if (line == null) more_data = false;
    else System.out.println (line);
}
catch (IOException e)
{ System.out.println ("IO error " + e) }
}
```

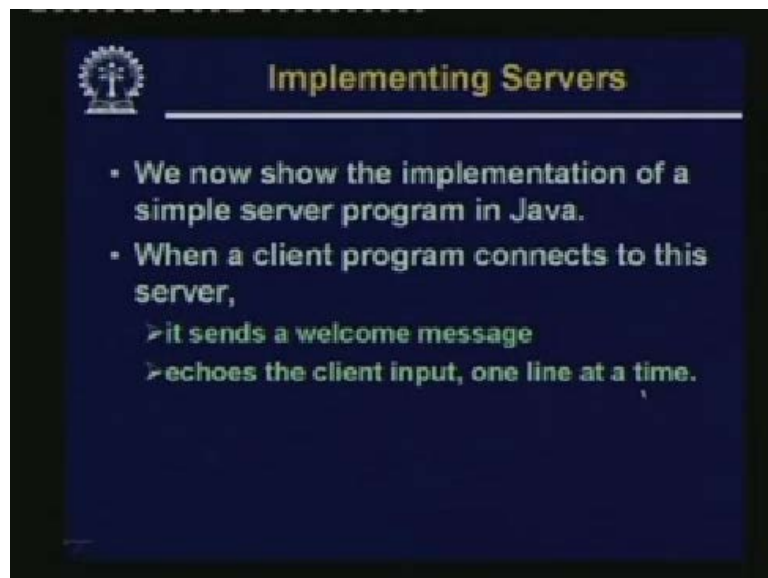
Then just to indicate we are outputting a string Established connection saying that connection has been established. Now the server is expected to send back some data to the client. So in this loop while more data, you continue doing this. You read a line of text using the readLine function. If the line is null which will mean that there is no more text to come. So if line is null you immediately set more_data to false. Otherwise you print this line on the screen, so basically in this file loop you are reading one line if the line is null you set the flag to false otherwise you print it. As soon as you set the flag to false next time the while loop will come out and the program will stop and if during transmission, there is some IO error. Then this exception handler will catch it and this IO error with the error code or message will be printed. So this particular example is a very simple illustration as to how we can write a client program which can be made to connect to a server and do certain things here. For example we have just **decode** the message to the screen whatever is coming back. Now a few things you should remember here when you are developing a network application.

(Refer Slide Time: 27:27)



That all networking code should be enclosed in the try catch block. This will make your application robust in the sense that in case of any error in the software during the transmission whatever kind of error may be encountered. Try catch block implicitly will have an exception handler so if there is an exception which is raised the exception handler can be used to print a suitable error message through which you can understand exactly what has happened otherwise it will not be possible for the user to know exactly why my application is not running properly what is the matter. So as I had said most of the network related methods. They throw the IO exception, this particular kind of exception whenever there is some error in process right.

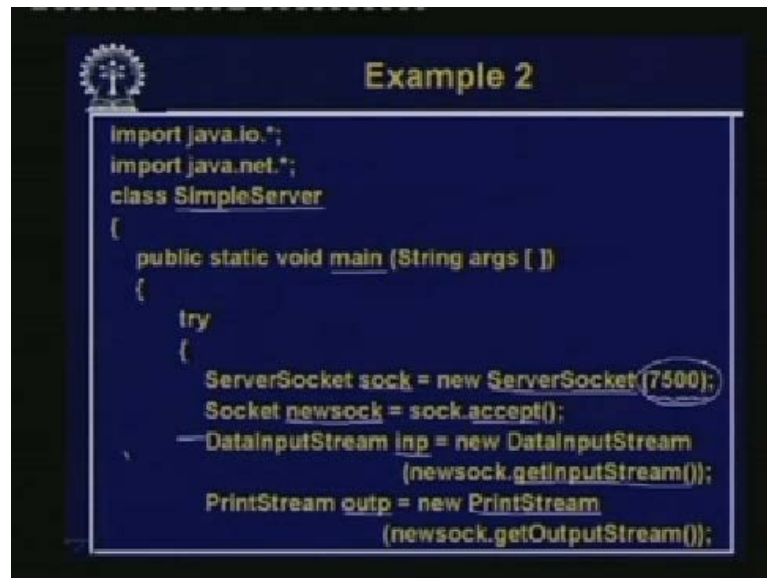
(Refer Slide Time: 28:29)



Now let us see how you can implement servers. The example we have shown that is primarily meant to implement a client. So as I had said earlier the client and server will be asymmetric. When you are writing a program there will be some subtle differences in the two the kind of

methods. We use the way you sequence them they will be different. We will see, so here the server that we write is as follows. The server you can actually use in conjunction in the client program you have just written. So this particular sever after it receives a request from a client. It will send back a welcome message to the client then it will expect the client to send some data. And whatever data it receives from the client it will simply echo it back one line at a time. So the server will be reading one line from the client it will again send it back, again read the next line again send it back this will be continued till the next line is null or some delimiter is set.

(Refer Slide Time: 29:35)



```
Example 2

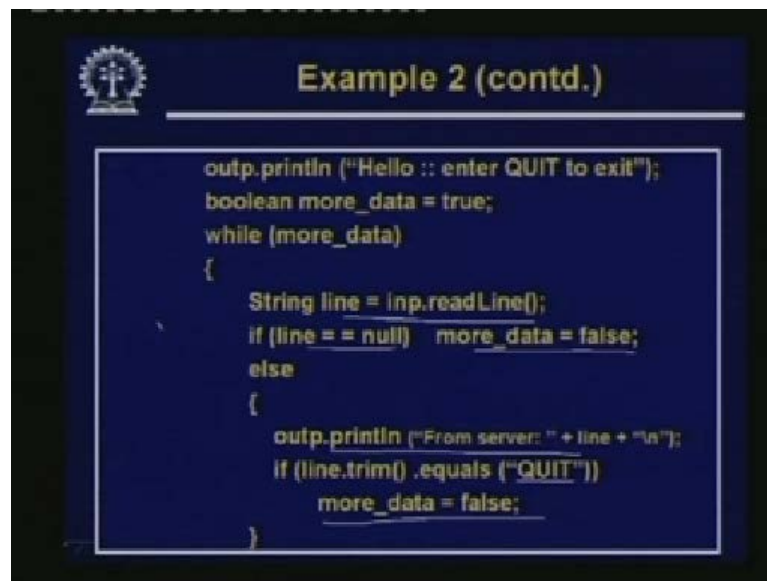
import java.io.*;
import java.net.*;
class SimpleServer
{
    public static void main (String args [ ])
    {
        try
        {
            ServerSocket sock = new ServerSocket(7500);
            Socket newsock = sock.accept();
            DataInputStream inp = new DataInputStream
                (newsock.getInputStream());
            PrintStream outp = new PrintStream
                (newsock.getOutputStream());
```

So this second example looks like this. So again as usual we have this java io, java net, this package is included. This is a simple server, this is the name of the class, and this again is an application with the main method. Now you see there some differences you will find between the client program and this particular server program. First thing is that for creating a socket you are using. A different socket called server socket in case of a client program you had used the class called simple socket, only socket. But here you using server socket, see server is not trying to connect to any machine. So you need specify any IP address, here for a server socket it takes only one parameter it is the port number on which the server should wait for a client request to arrive. So server socket has a single parameter only indicating the port number on which it should be continuously listening to. Then there is an accept method which is available under this server socket class. So sock dot accept if you call this will mean by default the server will be waiting.

This accepts is the function caller method which makes the server wait the server is waiting till a client request comes. As soon as it comes it will come out of accept otherwise it will get blocked at that statement itself. So after a client request has come only then the server will move on to the next line. So here the accept will be returning a new socket descriptor. This will contain the detailed information about the client request. So with this information the server creates an inputStream from this new sock and the object is called inp and since we will have to send it back, echo it, [31:59 word nor clear] whatever is coming here echoing it on the screen print screen. So there is a similar to data input stream Java has a class called print screen. We are using that with the getOutputStream method on the same socket. New

sock will create a print string object called outp. So inp and outp these are the two objects we have created.

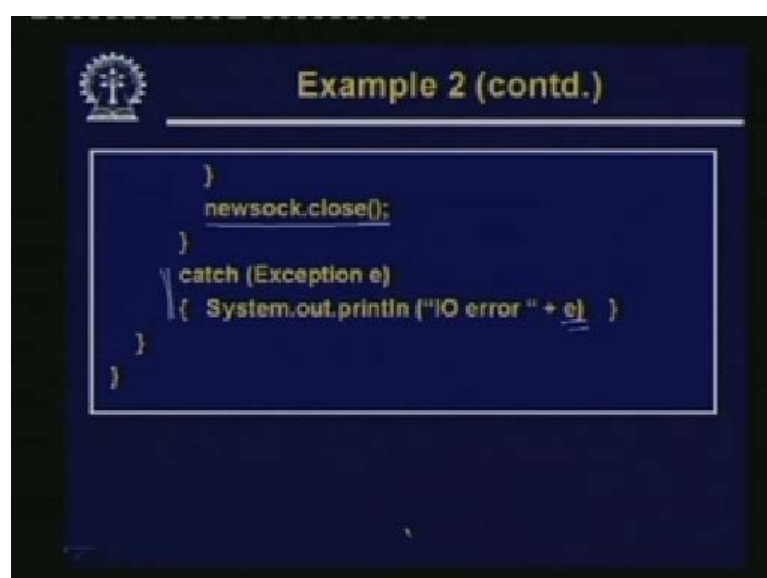
(Refer Slide Time: 32:25)



```
outp.println ("Hello :: enter QUIT to exit");
boolean more_data = true;
while (more_data)
{
    String line = inp.readLine();
    if (line == null) more_data = false;
    else
    {
        outp.println ("From server: " + line + "\n");
        if (line.trim().equals ("QUIT"))
            more_data = false;
    }
}
```

Then we are printing a welcome message in outp println “Hello enter QUIT to exit”. This is actually being outputted to the socket because outp is associated with that new sock. So it will not display on the screen, but on the socket. So the client can receive it on the other side. So the program is similar Boolean more data true. So means while this more data is not true, it reads a line of text. If it is a null its sets the flag to false, else it prints whatever it receives from the server which equals back. With this string from server attached to the line and a new line at the end you are checking after trimming after omitting the end of line or other. Or some other white spaces if it equals to QUIT. This means the last line of the text. So as soon as you get QUIT then also you can set the flag to false. So as soon as one of these two happens, this while loop breaks.

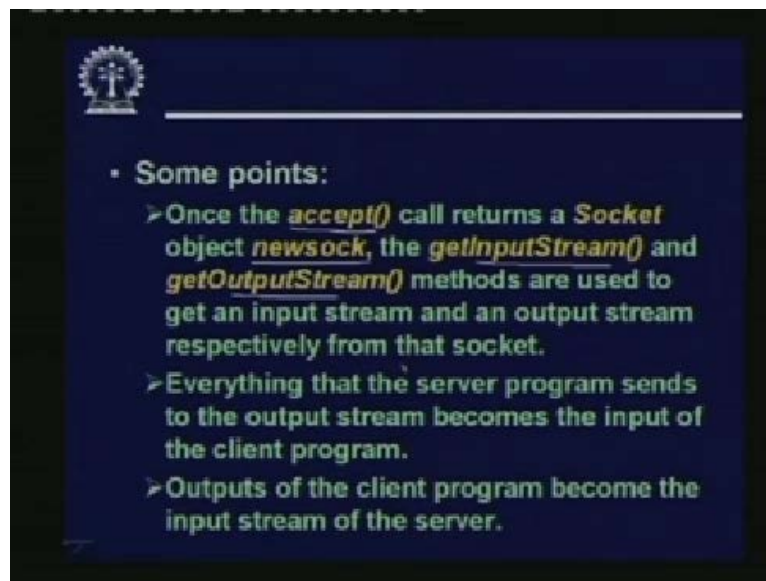
(Refer Slide Time: 33:36)



```
    }
    newsock.close();
}
catch (Exception e)
{ System.out.println ("IO error " + e) }
}
```

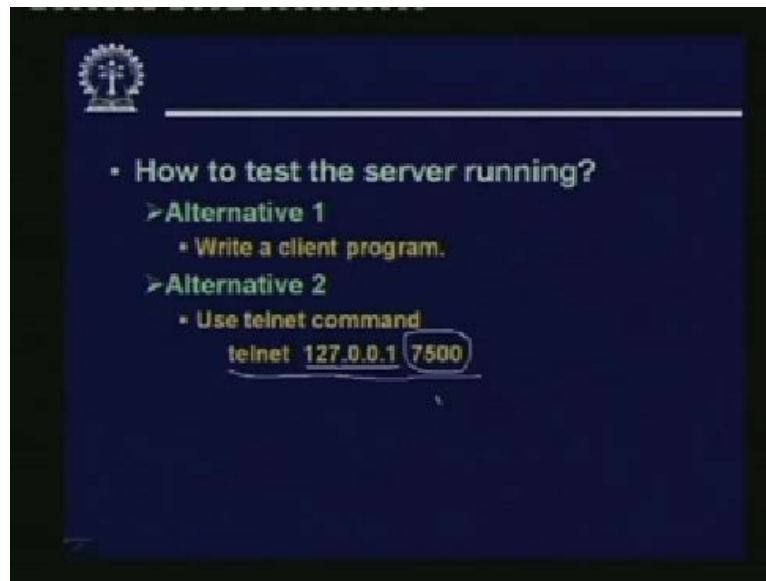
And when the while breaks at that point you can that newsocket, we just created and as usual the Exception handler is there. If there is some error in any one of the methods you are using. So exception will be caught and the error type that will be returned e will be displayed on the screen. So this example actually shows you how you can write a simple server program. Now in a sense the way you have written the server, this is an iterative server well of course we have not run the server in a loop. If we make a loop where the server after finishing a request again goes back to the accept to wait for the next client request. This will be an example of an iterative server. So only one client request can be handled at a time.

(Refer Slide Time: 34:33)



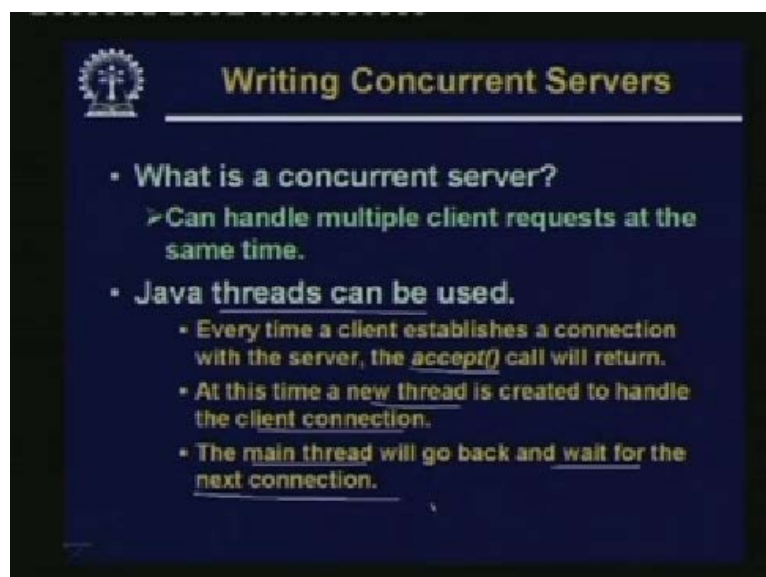
So, some points regarding the program out here, once the accept call is returned. So as the program shows it will be returning an object called newsock. After that on this newsock you can use the getInputStream and getOutputStream to perform read and write. And the way the program works is that everything that the server program sends to the output stream will become the input of the client. An output of client program will become input of the server. This is how the program was written.

(Refer Slide Time: 36:24)



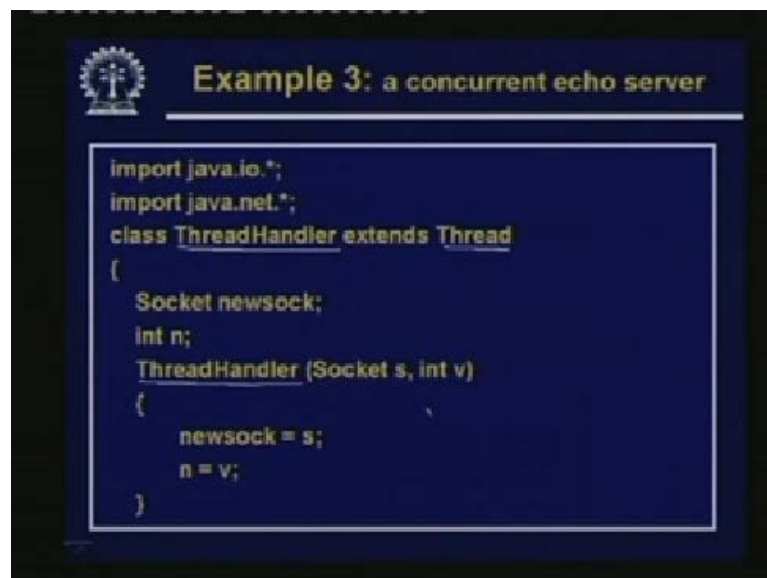
Now how to run this program? Run means how to test it. First of course will have to run the server. But to test the server, the first alternative is that you can write a client program specifically to server. Just like we have seen a earlier, how can write a client program and we can use this client program to connect to the server over that particular port no on which the server was written. And the accept call was return to wait for the particular port number. The second alternative is that, you know that the telnet command which is there. This can also be used to make a connection over a specified port number. So the telnet the first parameter will be the IP address or the name of the machine you are going to try to connect. And an optional second parameter is there where you can specify the port number. So if you connect the server like this, then what ever you type after the telnet will go to the server and whatever comes out from the server you can see on this screen. So this kind of a simple eco server you can very well test using the telnet command on the same machine are any other machine on the network right.

(Refer Slide Time: 36:44)



Now let us see how we can write a concurrent server which can handle several simultaneous client request at the same time. Now in java, there is no direct concept of creating processes because when you had earlier seen that how can do this thing in C or C plus plus kind of languages. There we have seen or we are told that whenever you are designing a concurrent server. Then we will separate copies of this server process that we created, there will be several processes which will be handling all the client request. But in java instead of process of what you create are called java threads. Java language supports this thread manipulation. You can create multiple threads which work similarly to process of course a thread is called a light weight process. The overhead of thread switching and thread handling is much less than that of a process. So for this purpose you can use Java threads. So the idea is that every time a client will make a connection request in the server. The accept call will return. So what this server will do? Now at this point this server will create a new thread and will ask the new thread to handle the client connection. However the main thread which was the original server program will go back to the accept call and wait for next connection to arrive. This is the basic idea behind how we can write such a concurrent server.

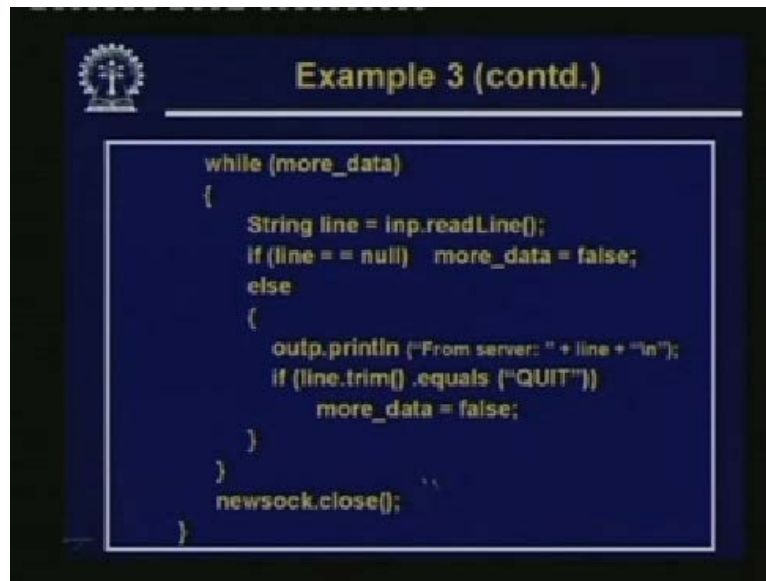
(Refer Slide Time: 38:34)



```
import java.io.*;
import java.net.*;
class ThreadHandler extends Thread
{
    Socket newsock;
    int n;
    ThreadHandler (Socket s, int v)
    {
        newsock = s;
        n = v;
    }
}
```

Let see the example that how we can do that. Here the name of the class we have given has ThreadHandler. Since it uses threads, that is why it extends or it is derived from a class thread since you are writing a server. We have to use the newsocket, we shall see it later. ThreadHandler, this is the constructor class of the ThreadHandler. This is a particular thread handling class. So the constructor says that whenever you created an instance of this thread. This constructor will get executed, this constructor takes two parameters. One is the socket on which the thread has to listen or has to communicate and second one is a integer number. But this is just a thread number which doubt. So there are some global variables newsock and n. So whatever is returned, here will be assigned to these variables newsock and n. So basically ThreadHandler have the purpose of initialising the variables newsock and n to a socket number and the thread number.

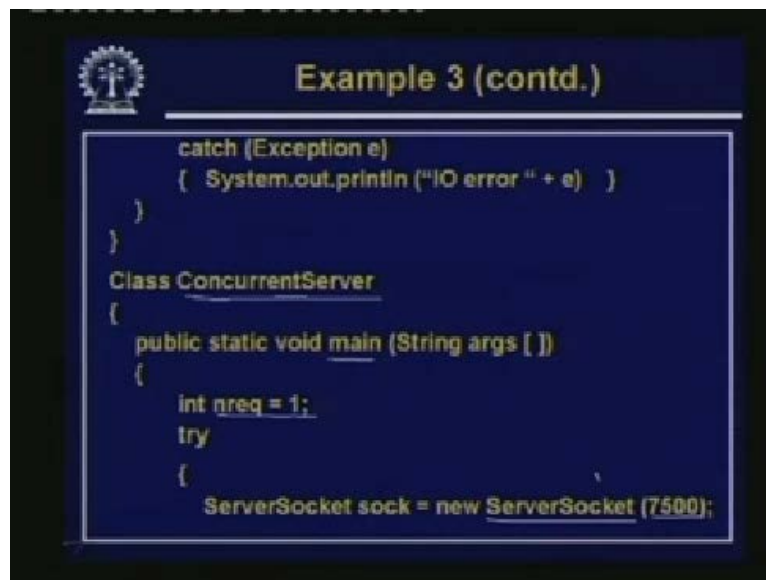
(Refer Slide Time: 40:00)



```
while (more_data)
{
    String line = inp.readLine();
    if (line == null) more_data = false;
    else
    {
        outp.println ("From server: " + line + "\n");
        if (line.trim().equals ("QUIT"))
            more_data = false;
    }
}
newsock.close();
}
```

This is the main method for the thread handling. Whenever you call it, this will be run. So here what it is doing? Using the DataInputStream again using this newsock, it is creating an input stream. The object name is inp. Similarly from that newsock again from getOutputStream, it is creating an object outp. Here it is first sending a welcome message just like that previous example. This will be going to this newsock. Then again this Boolean variable is set to true.

(Refer Slide Time: 40:40)

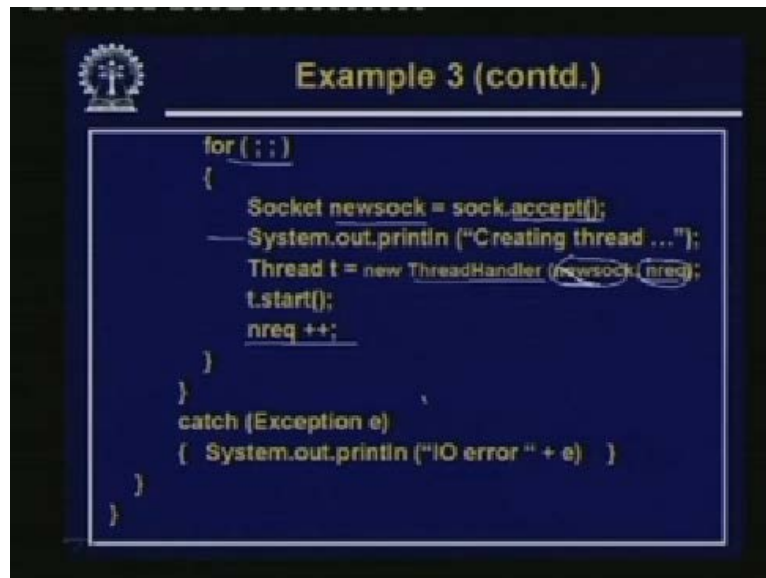


```
catch (Exception e)
{ System.out.println ("IO error " + e) }
}
}
Class ConcurrentServer
{
    public static void main (String args [ ])
    {
        int nreq = 1;
        try
        {
            ServerSocket sock = new ServerSocket (7500);
```

This part is the same as in the previous program. So actually what is happening here is that, the part of the program that you have seen so far, that is just the thread handling class. Whenever you create a thread, the thread will be opening an InputStream and OutputStream on the socket that you have passed as a parameter and it will be reading the data from the socket and outputting it back equating it. So this is exactly what every thread will do every time you create a thread. This part of the program is the same as the previous one the

remaining part. Then exception handler, now here comes the server class. The class that you have seen so far that is the ThreadHandler. But now is the actual server. There is a main method. So there is a variable which you initialize to one and since the server you are using the ServerSocket class to define a socket on number 7500 because all the clients will be sending the request over port number 7500. This is the assumption.

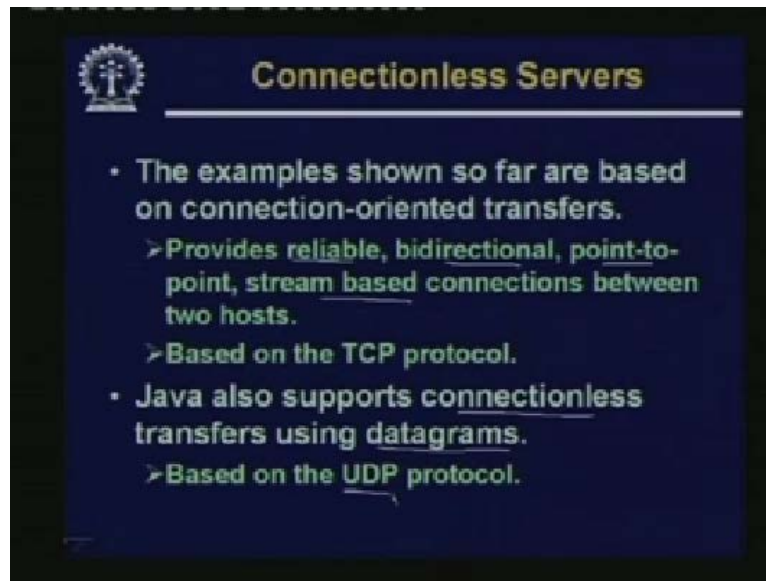
(Refer Slide Time: 41:58)



```
for (;;)
{
    Socket newsock = sock.accept();
    System.out.println ("Creating thread ...");
    Thread t = new ThreadHandler (newsock, nreq);
    t.start();
    nreq ++;
}
catch (Exception e)
{ System.out.println ("IO error " + e) }
```

Now in this loop out here you are calling exceptions. So as I said that this accepts us to call repeatedly, so in this infinite loop. The socket is called first and whenever there is a client request, it will come out to the next line. So a message will be printed creating thread, you are creating a new thread. By creating an instance of the ThreadHandler class you have just seen and as you can see the two parameters you are passing here are this newsock socket number out here. And this nreq which was initialized to one, so after starting this thread t dot start where the run method will be invoked. You increment nreq by one, so that next time whatever you create another thread that will have a number of two. So thread number is also mandatory.

(Refer Slide Time: 42:54)

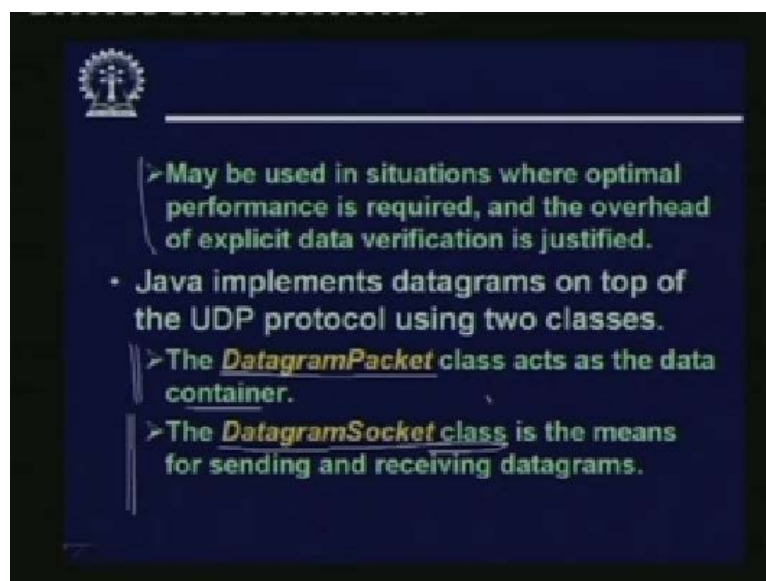


The slide features a dark blue background with a white logo in the top left corner. The title 'Connectionless Servers' is centered at the top in a yellow font. Below the title, there are two main bullet points in white text, each followed by a sub-bullet point in yellow text.

- The examples shown so far are based on connection-oriented transfers.
 - > Provides reliable, bidirectional, point-to-point, stream based connections between two hosts.
 - > Based on the TCP protocol.
- Java also supports connectionless transfers using datagrams.
 - > Based on the UDP protocol.

So this example shows how you can write a simple concurrent server. Now the examples we have seen so far, these are examples of so called connection oriented server where a client establish a connection. The server is waiting for request over a particular port. Whenever it comes, then they start communicating over that particular socket, this is representative of the TCP mode of communication. But in general we can also have this so called connectionless socket or connectionless communication where we are actually using the UDP protocol for carrying out the communication. So let us see here, so as I had said that the examples we had seen so far, they are basically based on TCP which are connection orientated. They provide reliable, bi directional, point to point and stream based connection. But now we will see that how we can have connectionless transfer which will be using datagram there is no concept of a connection and such communication will be based on the UDP protocol.

(Refer Slide Time: 44:25)

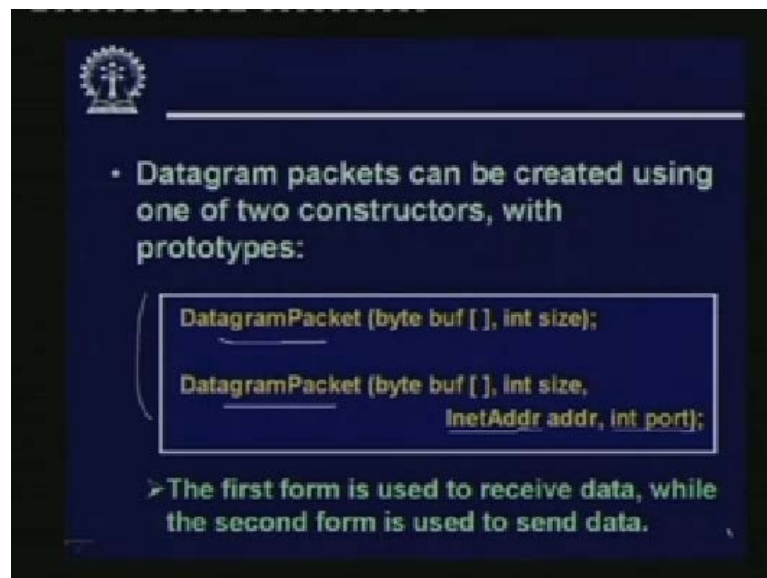


The slide features a dark blue background with a white logo in the top left corner. The text is presented in a list format with white and yellow text.

- > May be used in situations where optimal performance is required, and the overhead of explicit data verification is justified.
- Java implements datagrams on top of the UDP protocol using two classes.
 - > The DatagramPacket class acts as the data container.
 - > The DatagramSocket class is the means for sending and receiving datagrams.

Let us see as I had said before such communication using UDP protocol is typically used in situations where you need faster packet communication, where you want to reduce delay. And the overhead of explicit data verification in the application there is justified. Of course there is a second category of applications where you really do not care. If a few packets are corrupted or lost like some real time multimedia transfer over the net. Suppose you are receiving some voice packets over the network. Now even if one packet is corrupt, there is not much point in requesting the sender to again resend the packet and again play it from the beginning because that will cause an unwanted break in your voice playback. But rather you prefer that in one of the packet there will be small glitch in the voice. But the voice remains continuous. So this kind of application also you can use this UDP. Now in Java you can use UDP protocol using two classes DatagramPacket class and DatagramSocket class. In the DatagramSocket class you are actually creating socket that is the mains for sending and receiving datagrams. In the DatagramPacket class you are actually preparing the packet. That means you can call this as the data container. So the UDP packets are getting created here and the packets are sent or received.

(Refer Slide Time: 46:10)



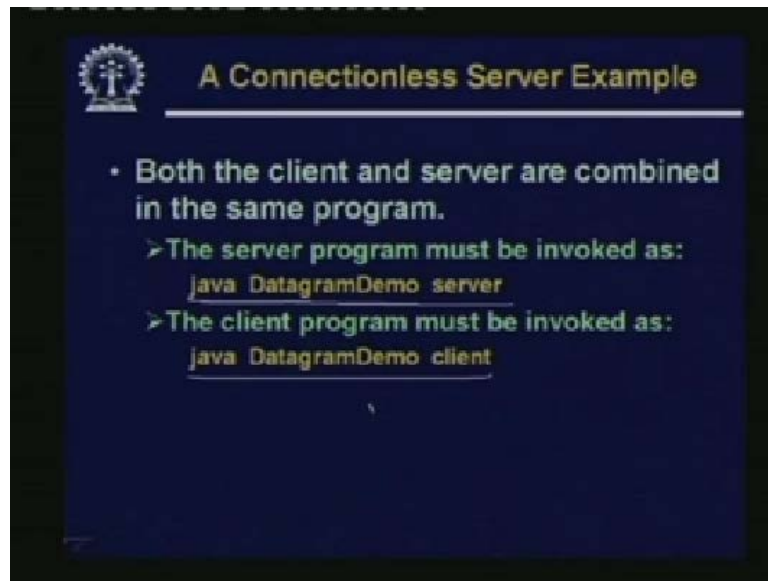
• Datagram packets can be created using one of two constructors, with prototypes:

```
DatagramPacket (byte buf [ ], int size);  
DatagramPacket (byte buf [ ], int size,  
                 InetAddress addr, int port);
```

➤ The first form is used to receive data, while the second form is used to send data.

So DatagramPacket can be created using the DatagramPacket constructor which has two different formats. In one you just specify a buffer and the number of bytes in the buffer size. So you want to receive data into this buffer whose maximum size is size in the other format. You specify buffer size; also you specify an IP address whose data type InetAddress and a port number. Now these two different forms exist because the first form you will use whenever you want to receive a packet. Because whenever you are wanting to receive a packet you need not have to specify the IP address of the other side because the packet will any way come to you. But when you are trying to send a DatagramPacket you need to specify where you want to send. So in the second form, you are explicitly specifying the addresses the address and port number. The first form is used to receive data and the second form is used to send data.

(Refer Slide Time: 47:29)



A Connectionless Server Example

- Both the client and server are combined in the same program.
 - The server program must be invoked as:
`java DatagramDemo server`
 - The client program must be invoked as:
`java DatagramDemo client`

So now let us see a simple connectionless example. Actually in this example, both the client and servers are together. So the program is written in such a way that the same program can be used as a client or as a server. When you want to use it as server, you should call it like this java name of the program server, if you want to use it as a client you should call it as java name of the program client. So the second parameter is a string which specifies in which mode you want to run this program right.

(Refer Slide Time: 48:47)



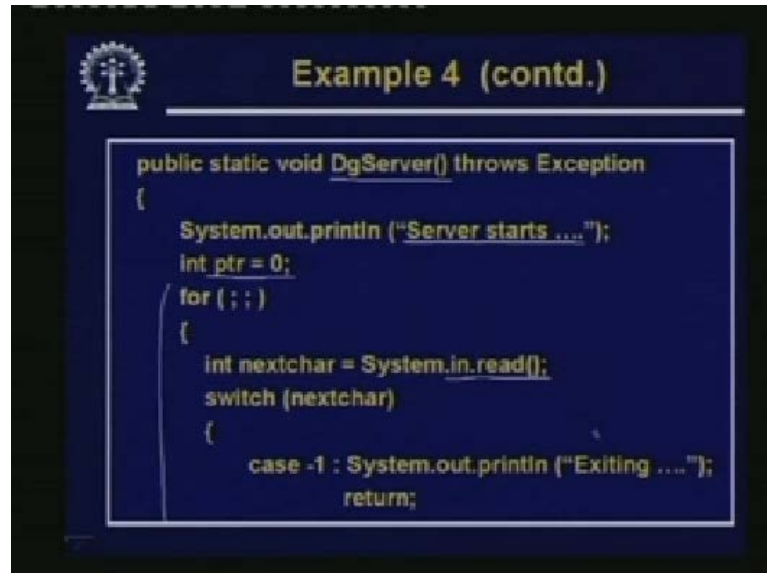
Example 4

```
import java.io.*;
import java.net.*;
class DatagramDemo
{
    public static int server_port = 7500;
    public static int client_port = 7501;
    public static DatagramSocket dgsock;
    public static byte buffer [] = new byte [512];
}
```

So now let us see how this program looks like. So as usual we have imported the io java dot io package. Java dot net package and the name of the class we have given as DatagramDemo. Now here since we are using it as a client and a server both we have predefined the port number to be used. So if it is a server then the server will be listening over port number 7500. If it is a client, then when the client is sending a packet to the server, it will be using port number 7501 as its own. Port number dgsock is a variable you are creating of type

DatagramSocket. And here you are defining a buffer where the data will be stored for transmitting or receiving. It is an array you are defining an array of bytes of maximum size 512.

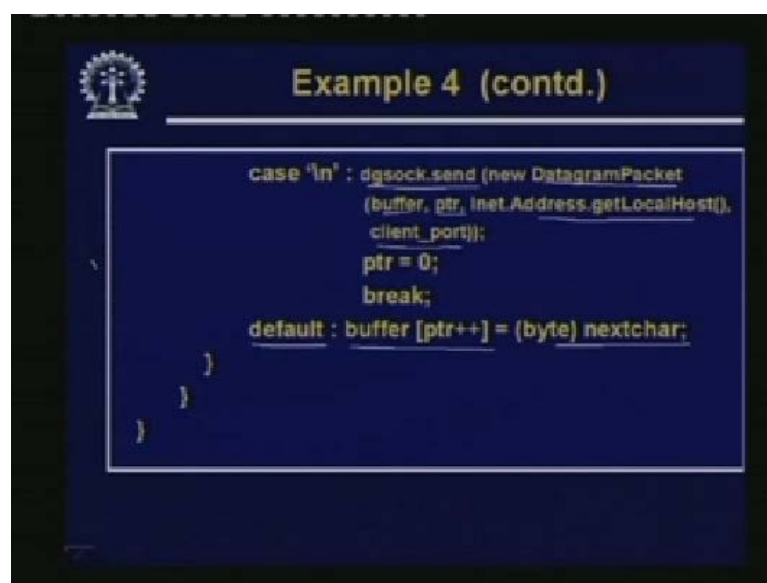
(Refer Slide Time: 49:14)



```
public static void DgServer() throws Exception
{
    System.out.println ("Server starts ....");
    int ptr = 0;
    for (;;)
    {
        int nextchar = System.in.read();
        switch (nextchar)
        {
            case -1 : System.out.println ("Exiting ....");
                    return;
        }
    }
}
```

Now here this method is the server DgServer method. So DgServer method will be invoked if it is, if you are running this in the server mode. So first it outputs a welcome message or just a message. The server starts ptr is a variable initialized to zero. In an infinite loop it runs. What it does it reads one character at a time; you see read line was the method which reads one line at a time in contrast. Read is a method which returns one character at a time. So you are reading one character at a time, then using a switch statement you are checking. If it is null it will return minus one, then you print Exiting and return.

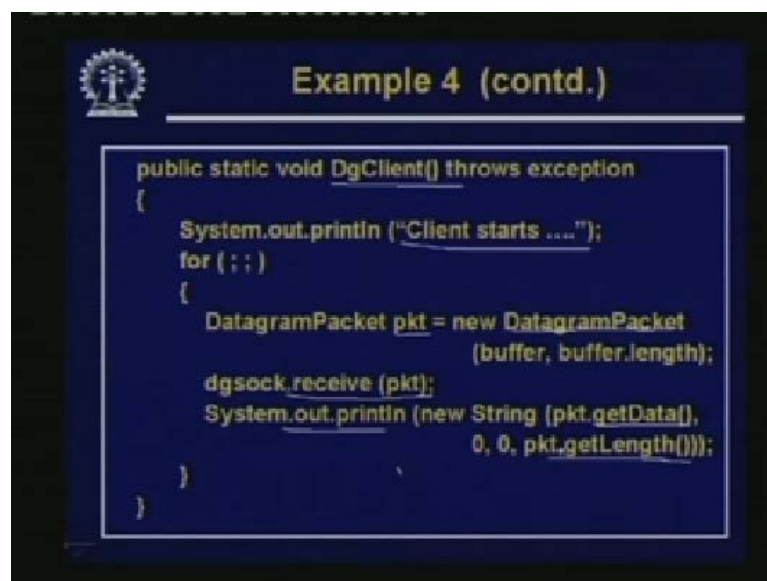
(Refer Slide Time: 50:15)



```
case 'in' : dgssock.send (new DatagramPacket
                    (buffer, ptr, InetAddress.getLocalHost(),
                    client_port));
            ptr = 0;
            break;
default : buffer [ptr++] = (byte) nextchar;
        }
    }
}
```


If it is a newline character that means the entire line is already transmitted to you. So now you are ready to send and in default case when it is not minus one, not newline, then you store the next byte which is coming into the buffer and increment pointer by one. But when you have encountered newline which means that the entire line has been received by the server. See the server, what it is doing? It is actually waiting and it is expecting some text to be received from the client. So it is reading character by character till the newline character is received. And as soon as it receives, it will echo it back to the client by explicitly sending a packet. So that is sending of the packet is done by the send method dgsock dot send. Here you are creating an instance of a DatagramPacket with the four parameters as I have said buffer pointer. The IP address of the local machine this you can get by using the getLocalHost method which returns the IP address of the local machine and the client port which we have initialized to 7501. So this is how the server works.

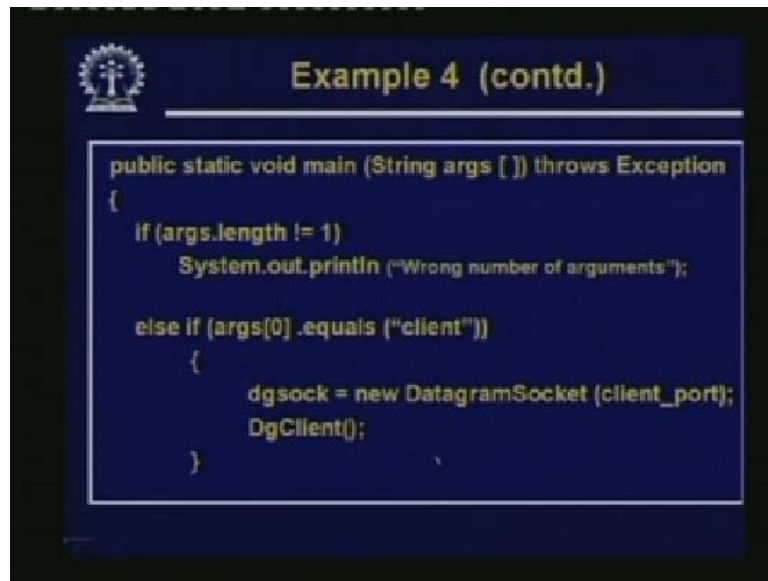
(Refer Slide Time: 51:45)



```
public static void DgClient() throws exception
{
    System.out.println ("Client starts ....");
    for (;;)
    {
        DatagramPacket pkt = new DatagramPacket
            (buffer, buffer.length);
        dgsock.receive (pkt);
        System.out.println (new String (pkt.getData(),
            0, 0, pkt.getLength()));
    }
}
```

Now the client. The client is another method whose name is DgClient. So again whenever the client starts you start by printing a message Client starts. Similarly you create a variable or an object, DatagramPacket object and you receive the packet from the server. You call dgsock dot receive to receive the packet which the server is sending and then you call System out println to print it. Of course so get you can get the data from the packet, packet dot getLength. So you know how many bytes or there. So you can use this to print the contents of this packet. So this is the client.

(Refer Slide Time: 52:47)



```
public static void main (String args [ ]) throws Exception
{
    if (args.length != 1)
        System.out.println ("Wrong number of arguments");

    else if (args[0] .equals ("client"))
    {
        dgsock = new DatagramSocket (client_port);
        DgClient();
    }
}
```

And this is the main program. Main program it checks the second argument, so it checks if the number of argument is not equal to one. Then it will say that is wrong number of arguments. Now args zero will be equal to the string that follows the name of the program. So it can either client or server, if it is client then it creates an instance of the DatagramSocket class and it calls DgClient.

(Refer Slide Time: 53:19)

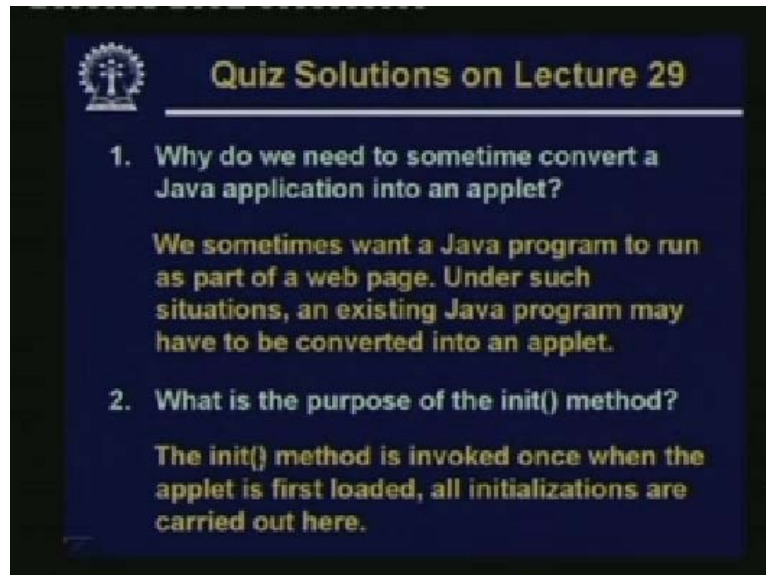


```
else if (args[0] .equals ("server"))
{
    dgsock = new DatagramSocket (server_port);
    DgServer();
}
}
```

And if it is the server, then it creates again DatagramSocket ad. It calls the DgServer. So this is how the client and server can be called from the main program depending on the second parameter that was. [53:40 word not clear] So with this actually we have come to the end of our discussion regarding how we can develop network applications in Java. Now here we had seen basically all the essential tools and techniques that we require in order to do that. We had seen how we can write a client program using connection orientated technology. We had seen how we can write a server program using connection orientated technology. Both the

alternatives we have seen the iterative version and also the concurrent version. Just to recall most of the server program that we see around us are based on the concurrent version. Because most of the services have some variability in terms of the time they take and in terms of the kind of response the client expects and lastly we have also seen how we can utilize the connectionless technology of datagrams to send and receive packets. So now let us look at the solutions of the quiz questions posed in our last lecture.

(Refer Slide Time: 54:55)

A slide titled "Quiz Solutions on Lecture 29" with a logo on the left. It contains two quiz questions and their solutions. Question 1 asks why we sometimes convert a Java application into an applet, and the solution explains that it's for running as part of a web page. Question 2 asks about the purpose of the init() method, and the solution states it's invoked once when the applet is first loaded for initializations.

Quiz Solutions on Lecture 29

- 1. Why do we need to sometime convert a Java application into an applet?**
We sometimes want a Java program to run as part of a web page. Under such situations, an existing Java program may have to be converted into an applet.
- 2. What is the purpose of the init() method?**
The init() method is invoked once when the applet is first loaded, all initializations are carried out here.

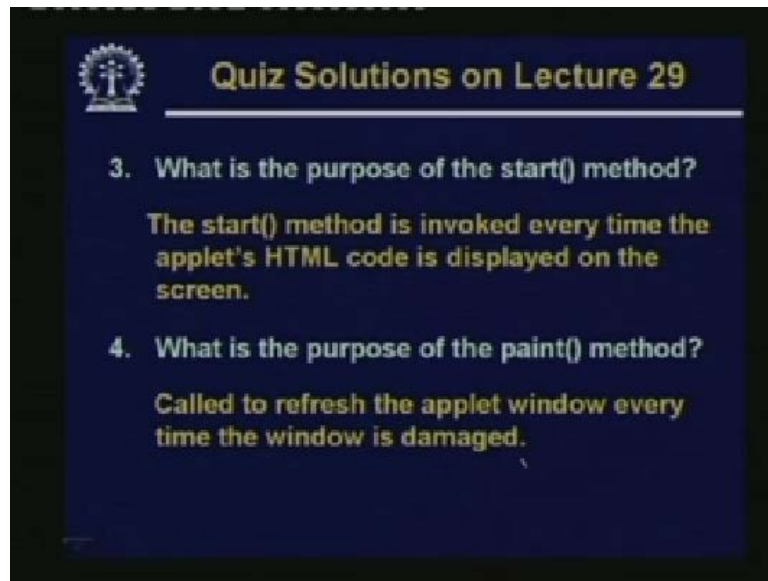
Why do we need to sometime convert a Java application into an applet?

Well we had seen applet is something which runs along with a web page. So if we want the Java program to be part of the web page. The existing Java program which you may have will have to be converted into an applet first.

What is purpose of the init method of course of an applet?

The init method is invoked once when the applet is first loaded, typically all the initializations that relate to the applet are carried out in the init method.

(Refer Slide Time: 55:34)



The slide features a dark blue background with a white logo in the top left corner. The title "Quiz Solutions on Lecture 29" is centered at the top in a yellow font. Below the title, there are two numbered questions and their corresponding answers, also in yellow text.

3. What is the purpose of the start() method?
The start() method is invoked every time the applet's HTML code is displayed on the screen.

4. What is the purpose of the paint() method?
Called to refresh the applet window every time the window is damaged.

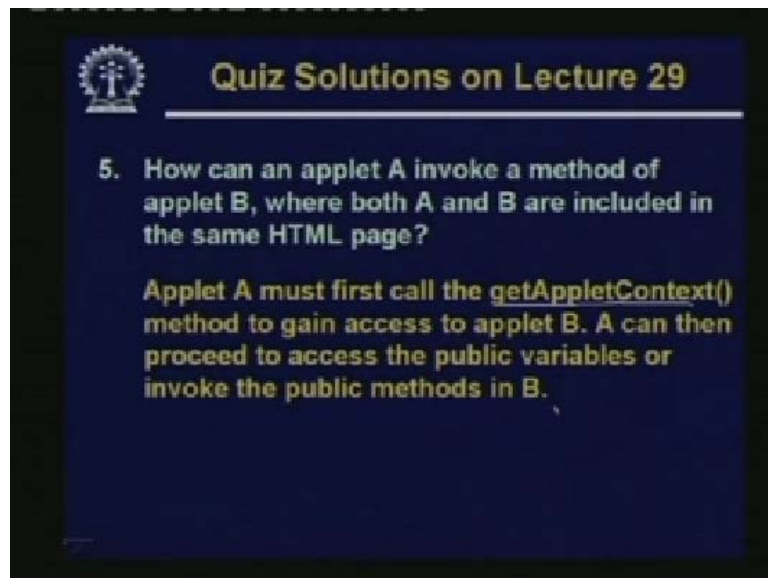
What is the purpose of the start method?

The start method is invoked every time the applet's HTML code is displayed on the screen.

What is the purpose of the paint method?

The paint method is called to refresh the applet window every time the window is damaged.

(Refer Slide Time: 55:56)

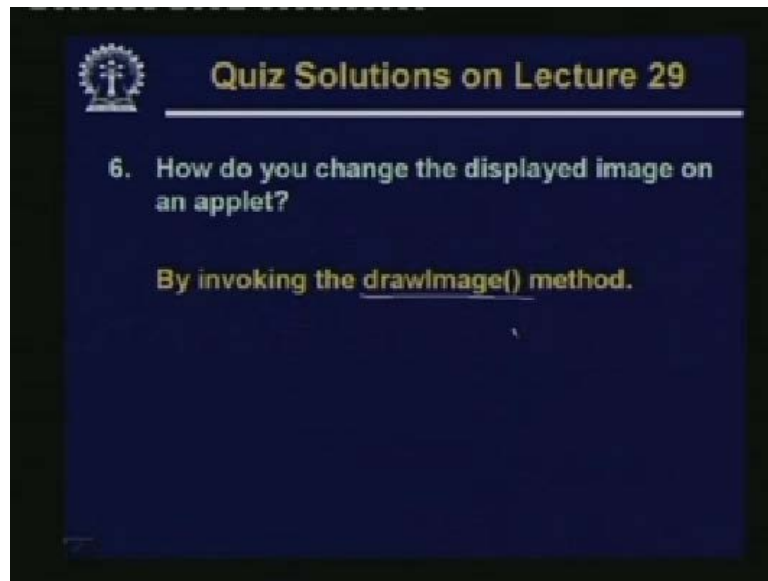


The slide features a dark blue background with a white logo in the top left corner. The title "Quiz Solutions on Lecture 29" is centered at the top in a yellow font. Below the title, there is one numbered question and its corresponding answer, also in yellow text.

5. How can an applet A invoke a method of applet B, where both A and B are included in the same HTML page?
Applet A must first call the `getAppletContext()` method to gain access to applet B. A can then proceed to access the public variables or invoke the public methods in B.

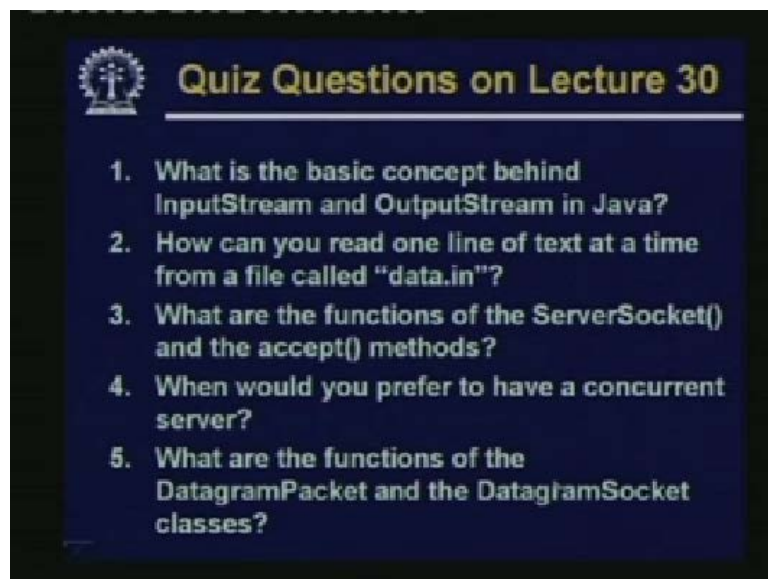
How can an applet A invoke a method of applet B, where both A and B are included in the same HTML page? Well applet A must first call the `getAppletContext` method to gain access to applet. A after that just using this method the object that will be returned. You can access all the public variables and methods available in B.

(Refer Slide Time: 56:26)



How can you change the displayed image on an applet? This you can do by calling the drawimage method. We had seen an example in our last class. So now some questions from today's class. Let us see.

(Refer Slide Time: 56:42)



What is the basic concept behind InputStream and OutputStream in Java?

How can you read one line of text at a time from a file called data dot in?

What are the functions of the ServerSocket and accept methods?

Why did you why do use them? When would you prefer to have a concurrent server?

What are the functions of the DatagramPacket and the DatagramSocket classes?

So with this we come to the end of today's lecture. From the next lecture we shall be devoting a few hours on talking about some issues related to network and system security. How we can make our system secure? What are the technologies behind it and so on. So we shall be starting discussion on security related issues from our next lecture. Thank you.

(Refer Slide Time: 57:47)



Preview of next Lecture.

(Refer Slide Time: 56:50)



Intranet, Extranet, Firewall

From today's lecture, we shall be starting some discussions on security issues in computer networks. Actually we shall be talking about a few things. First we shall look at some of the security infrastructures that we require. Then we shall look at some of the low level techniques and technologies that we need to know about. Then we shall look at some of the typical applications that have been devised to work on the internet to make transactions safe and secure. The first lecture of the series is titled intranet extranet firewall.

So we shall be looking into these three different kinds of things.