CHAPTER 8 : NETWORKING

Java Networking is a concept of connecting two or more node together so that we can share resources. Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Protocol : Protocol is a set of rules & regulation that governs the data transfer between two different nodes. Example: TCP, FTP, Telnet, SMTP, POP etc.

Port Number : The port number is a 16 bit no. that uniquely identifies an application on a computer. It acts as a communication endpoint between applications. The port number is associated with the IP address for communication between two applications. OS uses port no. from 1 to 1023 for well known services like HTTP runs on port no. 80, finger on 79, ftp on 21, telnet on 23,etc. The database, web server, etc. uses port no. from 1024 to 10000. Hence a user defined application should use port no. more than 10000 to avoid conflict.

Socket : A socket is an endpoint between bi directional communication channel that converts the data from physical medium to logical medium or from analog to digital and vice versa.

TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection. This is analogous to making a telephone call. If you want to speak to Aunt Baguli in Kendrapara, a connection is established when you dial her phone number and she answers. You send data back and forth over the connection by speaking to one another over the phone lines. Like the phone company, TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.

TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. The order in which the data is sent and received over the network is critical to the success of these applications. When HTTP is used to read from a URL, the data must be received in the order in which it was sent. Otherwise, you end up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called *datagrams*, from one application to another. Sending datagrams is much like sending a letter through the postal service: The order of delivery is not important and is not guaranteed, and each message is independent of any other.

For many applications, the guarantee of reliability is critical to the success of the transfer of information from one end of the connection to the other. However, other forms of communication don't require such strict standards. In fact, they may be slowed down by the extra overhead or the reliable connection may invalidate the service altogether.

Consider, for example, a clock server that sends the current time to its client when requested to do so. If the client misses a packet, it doesn't really make sense to resend it because the time will be incorrect when the client receives it on the second try. If the client makes two requests and receives packets from the server out of order, it doesn't really matter because the client can figure out that the packets are out of order and make another request. The reliability of TCP is unnecessary in this instance because it causes performance degradation and may hinder the usefulness of the service.

Another example of a service that doesn't need the guarantee of a reliable channel is the ping command. The purpose of the ping command is to test the communication between two programs over the network. In fact, ping needs to know about dropped or out-of-order packets to determine how good or bad the connection is. A reliable channel would invalidate this service altogether.

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data from one application to another. Sending datagrams is much like sending a letter through the mail service: The order of delivery is not important and is not guaranteed, and each message is independent of any others.

Ip Address :- It is a 32/128 bit number that uniquely identifies a computer across the internet/intranet.

IPV4 : It is a 32 bit IP Address that has 4 part and each part is 8 bit which can hold a value maximum upto 256.
Ex: 192.186.10.5

IPV6 : It is a 128 bit IP Address that has 8 part, but 6 part is considered as IP Address and each part is 16 bit which can hold a value maximum upto 65355. The last 2 part is considered as MAC address(MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.)
Ex: The older values can be represented as 192.186.10.5.0.0.0.0

InetAddress class: - This class is used to return an IP address of a Computer to a Java Program. This class has no constructor like Runtime class, But it has some static method by which we can get a IPaddress of a computer.

public static InetAddress getLocalHost() :- It will return an Ip Address of the machine on which this method will execute.
public static InetAddress getByName(String name) :- It will return the Ip Address of the computer whose name is given in the argument(eg.”lab5”,”192.168.10.5”etc).

Creating A Socket /Networking Application byusingTCP/IP: The java.net package has given the ServerSocket & Socket class by which we can create a networking application in java & protocol will become TCP/IP implicitly.

ServerSocket class:- This class must be used by a Java program which will behave like a Server.

ServerSocket(int portnumber):- This Constructor will find the Server application with the specified port number so that a socket will identify the server by the given port number. This Constructor does not allow the Server application to communicate with more than one Client.
ServerSocket(int portnumber, int backlog):- This Constructor is similar to the above Constructor but it allows a Server application to communicate with different clients maximum for backlog no of times. If we provide 5 as the 2nd argument then the server can connect 5 times with different clients but the communication will be made with the existing client.
public Socket accept() :- This method of the ServerSocket class is a blocking method, this will block the execution of the server Application. So that the server application will remain to listen the Client Connection Request.

Socket class :- This class is used by both server and client to transmit the Data.

Socket(InetAddress ob, int portnumber) :- This Constructor is a Connection request, from client to server. Hence the port number & IPAddress must belong to server. The port number & IPAddress of the client is not necessary for the server because the client will make the Connection request. Once the connection is made the communication can be started between the client and server. Hence port number and IPAddress of Socket object. The accept() that blocks the server application will make a connection & will return a Socket object to the server. In order to transfer data between Server & Client Server has to write the Data to its own Socket & the Client must read from its own Socket & vice versa.

public InputStream getInputStream() :- It is used to read some bytes from the Socket.
public OutputStream getOutputStream() :- It is used to write some bytes to the Socket.

Creating A Socket Application By Using UDP :-The DatagramPacket & Datagram Socket classes are used to create Socket Application in which the protocol will remain as UDP implicitly. The UDP is like Postal Service.

DtagamPacket class:- This class is used as a Container of data. It is like an Envelope.

DatagramPacket(byte b[], int length, InetAddress ob, int port) :- This constructor is used to bundle up the data, in order to send. It will collect the data from the 1st index of the byte array up to the length of the byte array as given in the 2nd argument. Since this DatagramPacket will be transmitted from the sender to receiver, Here the InetAddress and port number must belong to the receiver(as we are mentioning the address of the receiver).
DatagramPacket(byte b[],int offset , int length, InetAddress ob, int portno) :- It is similar to the above constructor but it will collect the data from byte array starting from the index given in 2nd argument up to the length given in the 3rd argument. It is also used to create a Datagrampacket in order to send.

DatagramPacket(byte b[], int length) :- This constructor is used to receive the data from the DatagramSocket. The receiver will collect the actual data from the byte array given in the above constructor, & the data is stored from the beginning of the byte array up to the length specified by the 2nd argument. When we provide the object created by this constructor to the DatagramSocket then the DatagramSocket will provide the data in DatagramPacket form, then this constructor will return the original data in the byte array to the java program.
DatagramPacket(byte b[],int offset , int length) :- It is similar to the above constructor by which we can recive the data but it will keep the data in byte array starting from the index of the byte Array as given in 2nd argument up to the length as specified in 3rd argument.

DatagramSocket class:-- This class is like a carrier(Transporter) of the data. It is responsible for to transmit the data from sender to receiver.
DatagramSocket(int port) :--- Since here is no connection between a sender & receiver, hence both application must be registered with different port number.

public void send(DatagramPacket dp)---This method will accept a DatagramPacket in order to send.
public void receive(DatagramPacket dp)---This method is used to receive the data in the form of a DatagramPacket. Hence it will accept a DatagramPacket in the argument.
Using TCP/IP:
Store the file as Server1.java :
```
import   java.io.*;
import   java.net.*;
class Server1 {
public  static void main(String args[])  throws Exception {
  ServerSocket ss= new ServerSocket(12345); /* 12345 is user defined portno. for Server1 appl^, so that a socket will identify this appl with 12345  portno. */
Socket  s = ss.accept(); /* Blocking the execution of Server, i.e. the Server should wait to get the connection request from Client*/
System.out.println("connected  to client");

 BufferedReader  br  =new  BufferedReader(new  InputStreamReader(s.getInputStream()));  /*  Creating  an object of BufferedReader, i.e. read pointer, by which we can read some bytes in terms of line from the Socket, i.e.from Client. */
System.out.println(br.readLine()); /* Reading a line/String from Client*/
br.close();
}};
```

Store the file as Client1.java :
```
import   java.io.*;
import   java.net.*;
```

```
class  Client1 {
public  static void main(String args[])  throws Exception {
Socket  c = new Socket(InetAddress.getLocalHost(), 12345); /*conn req. from client to server.If we keep the server in different computer then the 1st parameter should be InetAddress.getByName( "lab56" or"192.168.10.2") say lab3 or 192.168.10.2 is the name or ipaddress of different computer */
System.out.println("connected  to server");

PrintStream ps = new PrintStream(c.getOutputStream() , true); /* Creating an object of PrintStream by which we can write data to Socket*/
 ps.println("hello server"); /* writing a line to socket, i.e. for Server1*/
ps.close();
}}
```

In the above example the Client1.java is sending the data only once to the Server1.java program. Steps to execute the above program :-

Step 1 : Open two dos prompt, change to appropriate directory where the above program are stored.
Step 2 : type **java Server1** in one dos prompt to start Server1.
Step 3 : type **java Client1** in another dos prompt to start Client1, change to previous dos prompt where Server1 is running. You will get "hello server".

Now let us create another program where the Client will send multiple request and server will read multiple times.

Store the file as Server2.java :

```
import  java.net.*;
import  java.io.*;
class   Server2  {
  public  static void main(String args[]) throws Exception  {
  ServerSocket ss = new ServerSocket(12345);
    Socket  s  =  ss.accept();
   System.out.println("Connected to Clnt");

 BufferedReader br= new BufferedReader(new InputStreamReader(s.getInputStream()));
  PrintWriter  pw = new PrintWriter(s.getOutputStream(),  true);  /* br- read from socket, pw - write to socket */
String k=br.readLine(); //reading from socket
 while(!k.equals("exit")) {
System.out.println("from Clnt :" +k);
 StringBuffer sb = new StringBuffer(k);
  sb = sb.reverse();
  k = sb.toString();
  pw.println(k); //writing to socket
  k=br.readLine();
 }
  pw.close();
  br.close();
}};
```

Store the file as Client2.java :

```
import  java.net.*;
import  java.io.*;
class   Clnt2  {
  public  static void main(String args[]) throws Exception  {
  Socket c = new Socket( InetAddress.getLocalHost() , 12345);
  System.out.println("Connected to Serv");

 BufferedReader kb= new BufferedReader(new InputStreamReader(System.in));
  PrintWriter  pw = new PrintWriter(c.getOutputStream(),  true);
 BufferedReader br= new BufferedReader(new InputStreamReader(c.getInputStream()));
/* kb- to read from Keyboard, pw - to write on socket, br- to read from socket */
 String z=kb.readLine(); //reading from keybrd
  while( ! z.equals("exit")) {
      pw.println(z); //writing to socket
      System.out.println("from Serv : " +br.readLine()); //reading from  socket
      z=kb.readLine();  //reading again from keybrd
   }
  pw.println("exit"); //  pw.println(z);
  pw.close();
  br.close();
  kb.close();
}};
```

Lets us create a charting program which will contain GUI component.

Store the file as Serv3.java :

```
import  java.net.*;
import  java.io.*;
import  java.awt.*;
import  java.awt.event.*;
class  Serv3 extends  Frame implements ActionListener ,Runnable {
TextArea  ta;
TextField  t1;
Button b1;
BufferedReader   br;
PrintWriter  pw;
public  Serv3() {
  super("server");
  setFont(new Font("dfasd", Font.BOLD, 40));
  Panel p1= new Panel();
  p1.setBackground(Color.orange);
  p1.setLayout(new FlowLayout( FlowLayout.LEFT));
  p1.add(t1= new TextField(10));
  p1.add(b1=  new Button("send"));
  b1.addActionListener(this);
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
   System.exit(0);
  }});
  add("Center", ta= new TextArea(2,3));
  add("South", p1);
  setSize(500, 500);
  setVisible(true);
  try {
   ServerSocket ss=new ServerSocket(12345);
   Socket s= ss.accept();
pw=new PrintWriter(s.getOutputStream(),  true);
br= new BufferedReader(new InputStreamReader(s.getInputStream()));
    Thread  tt = new  Thread(this);
     tt.start();
  }catch(Exception e) { }
}
  public void actionPerformed(ActionEvent ae) {
  try {
      pw.println(t1.getText());
      t1.setText("");
  }catch(Exception e) { }
}
  public void  run() {
    while(true) {
      try {
        ta.append(br.readLine()+"\n");
      }catch(Exception e) { }
    }
  }
  public  static void main(String args[]) throws Exception  {
   new   Serv3();
}};
```

Store the file as Clnt3 .java :

```
import  java.net.*;
import  java.io.*;
import  java.awt.*;
import  java.awt.event.*;
class  Clnt3 extends  Frame implements ActionListener ,Runnable  {
TextArea  ta;
TextField  t1;
Button b1;
BufferedReader   br;
PrintWriter  pw;
public  Clnt3() {
  super("client");
   setFont(new Font("dfasd", Font.BOLD, 40));
  Panel p1= new Panel();
  p1.setBackground(Color.orange);
  p1.setLayout(new FlowLayout( FlowLayout.LEFT));
  p1.add(t1= new TextField(10));
  p1.add(b1=  new Button("send"));
  b1.addActionListener(this);
  addWindowListener(new WindowAdapter() {
  public void windowClosing(WindowEvent we) {
   System.exit(0);
  }});
  add("Center", ta= new TextArea(2,3));
  add("South", p1);
  setSize(500, 500);
  setVisible(true);
  try {
  Socket c = new Socket( InetAddress.getLocalHost() , 12345);
pw=new PrintWriter(c.getOutputStream(),  true);
br= new BufferedReader(new InputStreamReader(c.getInputStream()));
  Thread  tt = new  Thread(this);
   tt.start();
  }catch(Exception e) { }
 }
  public void actionPerformed(ActionEvent ae) {
  try {
      pw.println(t1.getText());
      t1.setText("");
  }catch(Exception e) { }
}
  public void  run() {
```

```
  while(true) {
  try {
     ta.append(br.readLine()+"\n");
  }catch(Exception e) { }
  }
}
  public  static void main(String args[]) throws Exception  {
   new   Clnt3();
}};
```

Using UDP :

<u>Store the file as UdpServer.java :</u>

```
import  java.net.*;
class UdpServer {
 public static void main(String args[]) throws Exception {
   DatagramSocket ds = new DatagramSocket(12345);/*123 45 is the portno. of UdpServ*/
  byte  b[]= new byte[100];
  DatagramPacket dp= new DatagramPacket(b, b.length); /* dp is created to receive the data from the ds */
  ds.receive(dp); /* datagram socket is receiveing datagram packet */
String s =new String(b);
System.out.println("from client : "+s);
}};
```

<u>Store the file as UdpClient.java :</u>

```
import  java.net.*;
class UdpClient {
 public static void main(String args[]) throws Exception {
   DatagramSocket ds = new DatagramSocket(65432);/*654 32 is the portno. of UdpClnt*/
 String s1= "Hello  Mr.  Server how  are  you";
  byte  b[]= s1.getBytes();
  DatagramPacket dp= new DatagramPacket(b, b.length, InetAddress.getLocalHost(), 12345);/*dp is create to
send*/
  ds.send(dp); /* datagram socket is sending the datagram packet*/
```

URL class:- This class is used to hold the path of a resource, i.e. the object of this class will become a pointer to the resource. The resource may be a web site or file on the internet/intranet.

<u>openConnection() :-</u> This method of URL class will establish a connection between the java program and the resource, and it returns an object of URLConnection class, so that we can access the resource from the java program.

A URL(http://tomcat.apache.org/index.html) contains many information:

1. Protocol: In this case, http is the protocol.
2. Server name or IP Address: In this case, tomcat.apache.org is the server name.
3. Port Number: It is an optional attribute. If we write http// tomcat.apache.org:80/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. File Name or directory name: In this case, index.jsp is the file name.

//URLDemo.java

```
import java.io.*;
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
   URL url=new URL("http://tomcat.apache.org/index.html");
   System.out.println("Protocol: "+url.getProtocol());
   System.out.println("Host Name: "+url.getHost());
   System.out.println("Port Number: "+url.getPort());
```

```
  System.out.println("File Name: "+url.getFile());
}catch(Exception e){
   System.out.println(e);
}
}}
```

Methods of URL class	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

URLConnection class :- This class represents a communication link between the java program and the resource, so that we can either read or write data on the resource from the java program by the following methods.
public InputStream getInputStream() :- This method will read some bytes from the resource to the java program.
public OutputStream getOutputStream() :- This method will write some bytes to the resource from the java program.

URLConnection class :-

```
import  java.net.*;
import  java.io.*;
class URLRead {
  public static void main(String args[]) throws  Exception {
     //  URL  ur = new URL("file://d://nets//URLRead.java");
     URL ur = new URL("http://tomcat.apache.org/index.html");
     //URL ur = new URL("http://www.anything.com/index.html");
     URLConnection uc= ur.openConnection();
     BufferedReader br = new BufferedReader(new InputStreamReader( uc.getInputStream()));
     for(int i=0 ; i<5 ;i++)
        System.out.println(br.readLine());
     br.close();
 }};
```

HttpURLConnection class : The Java HttpURLConnection class is http specific URLConnection. It works for HTTP protocol only. By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc. The HttpURLConnection is subclass of URLConnection class.

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
  try{
      URL url=new URL("http://tomcat.apache.org/index.html ");
      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);
  }
}}
```