CHAPTER -10

JAVA SOCKET PROGRAMMING

THE CLIENT SERVER MODEL

- The Client /Server model is an application development architecture designed to separate the presentation of data from its internal processing.

- The Client requests for services and Server services these requests.

- The requests are transferred from the client to the server over the network.

- The processing that is done by the server is hidden from the client.

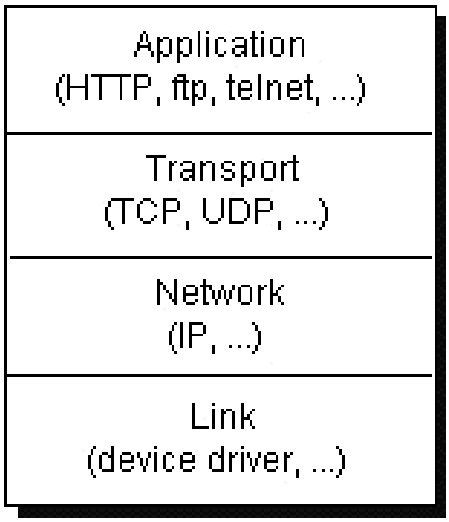
- The WWW provides an excellent Example

- The Server portion of Client/Server application manages the resources shared among multiple users who access the server through multiple clients.

PROTOCOL

- Network Protocols are set of rules and conventions followed by system that communicate over a Network.

- Examples : TCP/IP,UDP,NETBEUI



- Java Provides a rich library of network enabled classes that allow applications to readily access network resources.

- There are two tools available in Java for communication

    - Socket -> Uses Transmission Control Protocol
    - Datagram->Uses User Datagram Protocol.

TCP: [Transport Control Protocol]

- TCP is a connection-based protocol that provides a reliable flow of data between two computers.

- 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.

- 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.

- This is analogous to making a telephone call..

UDP: [USER DATAGRAM PROTOCOL]

- UDP 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.

- It sends independent packets of data, called *datagrams*, from one application to another.

- A datagram packet is an array of bytes sent from one program(sending program) to another(receiving program).

- As datagram follow UDP, there is no guarantee that the data packet sent will reach it's destination.

- 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.

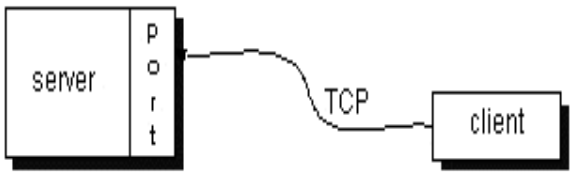
IP ADDRESSES AND PORT

- Data transmitted over the Network is accompanied by addressing information that identifies the computer and the port for which it is destined.

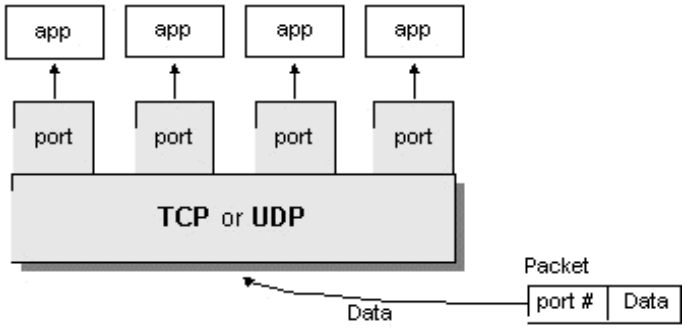
- The computer is identified by its 32-bit IP address

- Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.

- In connection-based communication such as TCP, a server application binds a socket to a specific port number.



- In UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application.



SOCKETS

- A socket is one endpoint of a two-way communication link between two programs running on the network.

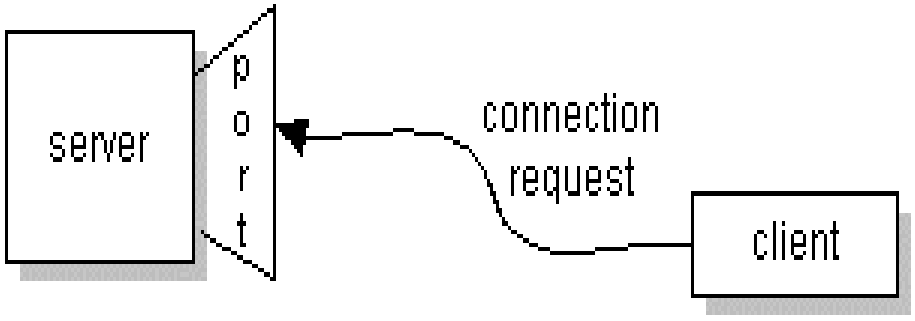
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

- Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.

- The server just waits, listening to the socket for a client to make a connection request.

- On the client-side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected.

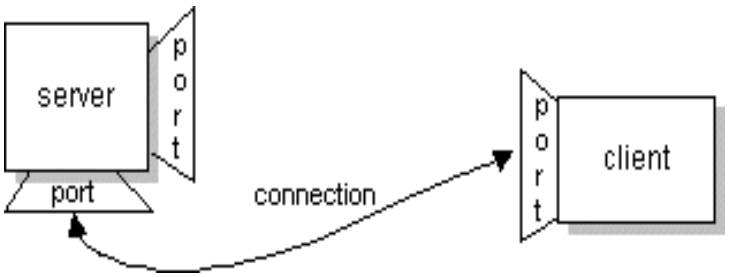
- To make a connection request, the client tries to make contact with the server on the server's machine and port.



- If everything goes well, the server accepts the connection.

- Upon acceptance, the server gets a new socket bound to a different port.

- It needs a new socket (and consequently a different port number) so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



- The java.net package in the Java platform provides a different classes associated with Sockets.

- **Socket class**

- It Provides methods for Stream I/O which makes reading from and writing to socket easy.

- **ServerSocket class**

- It is used for listening and accepting clients requests.

- It does not actually perform the service; instead it creates a socket object on behalf of the client.

SOCKET PROGRAMMING

1. CLIENT SIDE PROGRAMMING

2. SERVER SIDE PROGRAMMING

CLIENT SIDE PROGRAMMING

1.CREATING A CLIENT SOCKET

- The first step to create a socket client is to create a object of a Socket class.
- The constructor of the socket class takes 2 parameters
- `Socket(IP Address/HostName ,Port Number)`
- `EXAMPLE`

```
//----------------------------------------
   Socket client;

    try{
    client = new Socket("127.0.0.1",5001);
     }

    catch(Exception ae1)
     {
      //Unable to Create Socket
      }

//----------------------------------------
```

2.READING AND WRITING TO SOCKET

- Reading from and writing to a socket is similar to reading from and writing to files.
- PrintStream and BufferedReader classes are used
- PrintStream class is used to write to the socket.
- BufferedReader class is used to read from the socket.

```
//----------------------------------------

 PrintStream out =null;

 BufferedReader in = null;

 out=new
 PrintStream(client.getOutputStream());

 in=new
 BufferedReader(new
 InputStreamReader(client.getInputStream());

//----------------------------------------
```

- The getInputStream() and the getOutputStream() methods of the Socket class enables a client to communicate with the server.
- The getInputStream() is used for reading from the Socket.
- The getOutputStream() method is used for writing to a Socket.

- Declare another object of BufferedReader class to associate with the standard input so that data entered at the client can be sent to Socket.

```
//----------------------------------------

 BufferedReader stdin = new
            BufferedReader(new
            InputStreamReader(System.in));


 String str=stdin.readLine();

//----------------------------------------
```

3. CLOSING THE CONNECTION

- The statements given below closes the streams and the connection to the server.

```
//----------------------------------------

 out.close(); //close PrintStream

 in.close(); //Close BufferedReader

 stdin.close();

//----------------------------------------
```

SERVER SIDE PROGRAMMING

1. CREATING SERVER SOCKET

- To create a Server Socket,you need to create the object of ServerSocket class.
- When it recognizes a valid request, the Server Socket obtains the Socket object created by the client.
- ServerSocket class waits for the requests to come in over the network.
- It performs operations based on a request and returns the result to the client.
- The ServerSocket class provides constructor to create a socket on a specified port.
- **ServerSocket(Port No)**

```
//----------------------------------------
try{
ServerSocket server=new ServerSocket(5001);
}

catch(Exception ae1)
{
 // Cannot Start the Server
}
//----------------------------------------
```

2.LISTENING FOR CLIENT'S REQUEST

- It is provided with run method.
- In this case, the Server goes into an infinite loop and listens for clients request.
- When the server secures a connection form client,the accept() method of the ServerSocket class accepts the connection.
- The Server creates an object of the user-defined class Connection for the client,passing a Socket object to its constructor.
- Communication between the Client and Server occurs through this Socket.

```
//-----------------------------------------

public void run()
{
 try{
     while(true)
      {
        Socket client =server.accept();
        Connection conn=new
                        Connection(client);
      }
    }
 catch(Exception ae2)
 { //Server not listening; }

 } // End of run method
//-----------------------------------------
```

3.STARTING THE SERVER

- Server is started in main method.

- **EXAMPLE**

```
//-----------------------------------------

public static void main(String args[])
{
 new Server();
 }

//-----------------------------------------
```

CLIENT/SERVER WITH DATAGRAMS

- Connectionless transmission with datagrams is mor like the way mail is carried via postal service.

- DatagramSocket Class and DatagramPacket class is used in this architecture.

SERVER CONFIGURATION

1. CREATE DATAGRAM SOCKET

- Datagram Socket is created for sending and receiving packets.

- Port of a server is defined in the case of Server configuration.

```
//----------------------------------------

DatagramSocket server;
public Server()
{
    try
    {
        server=new DatagramSocket(5000);
     }

    catch(SocketException sck)
    {
     //Problem in Datagram Socket
    }
}

//----------------------------------------
```

2. RECEIVING PACKETS

- Setup the size of packets.

```
//----------------------------------------

  byte data[] = new byte[100];

//----------------------------------------
```

- The size of packet is allocated as 100 bytes.

- Create the object of DatagramPacket class and pass the name and length of packet in it's constructor.

```
//----------------------------------------

  DatagramPacket receivePacket = new
          DatagramPacket(data,data.length);

//----------------------------------------
```

- Receive the packet with the help of receive method of DatagramSocket class.

```
//----------------------------------------

  server.receive(receivePacket);

//----------------------------------------
```

- Different methods associated with DatagramPacket class are as follows.

METHODS	DESCRIPTION
getAddress()	Gets the IP Address of Sending Host.
getPort()	Gets the Port of Sending Host.
getLength()	Gets the Length of Data.
getData()	Gets the Data.

- All the methods should be used with the objects of DatagramPakect class.

```
//----------------------------------------

    String ipaddr=receivePacket.getAddress();

    String msg=new
                String(receivePacket.getData(),
                0,receivePacket.getLength());

//----------------------------------------
```

EXAMPLE TO VERIFY RECEVING PACKETS

```
//----------------------------------------

   public void waitForPackets() {

    while(true)
    {
        try {
             //Code to receive packets

             }

        catch(Exception ex1)
            {
             //Display message
            }
    }
  }

//----------------------------------------
```

3.SENDING PACKETS FROM SERVER

- Convert the message that is to be transmitted into bytes.

```
//----------------------------------------

  String msg ;
  byte data[]= msg.getBytes();

//----------------------------------------
```

- Create the object of DataPacket class and pass Four parameter in it's constructor.
- Four Parameters are
    - Byte array to send.
    - No of bytes to send.
    - IP Address of client.
    - Port where the client is waiting.

```
//----------------------------------------

    DatagramPacket sendPacket=new
        DatagramPacket(data,data.length,
        receivePacket.getAddress,
        receivePacket.getPort());

//----------------------------------------
```

- Send the packet with the help of send method of DatagramPacket class.

```
//----------------------------------------

    server.send(sendPacket);

//----------------------------------------
```

EXAMPLE TO VERIFY SENDING PACKETS

```
//---------------------------------------

try {
     String msg=//get Data;

     byte data[]=msg.getBytes;

     DatagramPacket sendPacket=new
     DatagramPacket(data,data.length,
        receivePacket.getAddress,
        receivePacket.getPort());

     server.send(sendPacket);

    }

catch(Exception ex1)
    {
    //Display Cannot Send
    System.exit(1);
    }

//---------------------------------------
```

CLIENT CONFIGURATION

1.CREATE DATAGRAM SOCKET

- Datagram Socket is created for sending and receiving packets.
- Port of a client is defined in the case of client configuration.
- DatagramSocket class is used.

```
//----------------------------------------

    DatagramSocket client;
    try
    {
        client = new DatagramSocket(4000);
    }
    catch(Exception excp1)
    {
        //Display Message
        system.exit(1);
    }

//----------------------------------------
```

2.RECEIVING PACKETS

- The size of packet is to be allocated.
- The object of DatagramPacket class is used along with two parameters : **Byte array to send** and **no of bytes to send.**
- send() method of DatagramSocket is used.

```
//----------------------------------------
    byte data[]= new byte[100];

    DatagramPacket receivePacket= new
        DatagramPacket(data,data.length);

    client.receive(receivePacket);

//----------------------------------------
```

3.SENDING PACKETS

- String message is converted to bytes.
- Object of DatagramPacket is used along with four parameters
- send() method of DatagramSocket class is invoked to send the required data packet.

```
//-------------------------------------

    String msg=//get message

    byte data[]=msg.getBytes();

    DatagramPacket sendPacket = new
    DatagramPacket(data,data.length,
    InetAddress.getLocalHost(),5000);

    client.receive(sendPacket);

//-------------------------------------
```

EXAMPLE OF SOCKET PROGRAMMING

CLIENT SIDE PROGRAMMING

```
//---------------------------------------
//Client.java

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

class Client
{
    public static void main(String args[])
                              throws IOException
    {
        Socket client;
        PrintStream out=null;
        BufferedReader in=null;
        BufferedReader stdin;

        try
       {
        client=new Socket("127.0.0.1",5001);

       out=new
      PrintStream(client.getOutputStream());

      in=new BufferedReader(new
                  InputStreamReader
                  (client.getInputStream()));
        }

        catch(Exception exc1)
        {
        System.out.println("Cannot Create
         Socket");
        System.exit(1);
        }

        stdin=new BufferedReader(new
              InputStreamReader(System.in));
```

```
        //Read Message From Server
        String msg= in.readLine();
        System.out.println(msg);
        System.out.println();

        System.out.print("Enter Radius :");
        String radius = stdin.readLine();

        //Write Radius to Socket
        out.println(radius);

        System.out.println("Writing Radius to
        Socket "+radius);

        //Read Area From Socket
        String area = in.readLine();
        System.out.println("Receiving
        Area...."+ area);

        out.close();
        in.close();
        stdin.close();
    }
}

//---------------------------------------
```

SERVER SIDE PROGRAMMING

```
//---------------------------------------
//Server.java

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

class Server extends Thread
{
    ServerSocket serverSocket;

    public Server()
    {
        try{
            serverSocket=new
                        ServerSocket(5001);
            }

        catch(Exception ex2)
        {
        System.out.println("Server Cannot be
                            Started....");
        System.exit(1);

        }

        System.out.println("Server Started
        Successfully..... !");

        System.out.println("Ready to Run the
        Clients.............!");

        System.out.println();

        this.start();

    }
```

```
    public void run()
    {
        try{
            while(true)
            {
            Socket client =
                        serverSocket.accept();
            Connection conn = new
                        Connection(client);
            }
        }

        catch(Exception ex3)
        {
            System.out.println("Server Not
             Listening.........!");

            System.exit(1);
        }
    }

    public static void main(String args[])
    {
        new Server();
    }
} // end of class Server

//---------------------------------------

class Connection extends Thread
{
    Socket netClient;
    BufferedReader fromClient;
    PrintStream toClient;

public Connection(Socket tempclient)
{
    netClient=tempclient;
```

```
    try
    {
        fromClient = new BufferedReader(new
                InputStreamReader
                (netClient.getInputStream()));

        toClient= new PrintStream
                (netClient.getOutputStream());
    }

    catch(IOException aex)
    {
        try{
            netClient.close();
        }
        catch(IOException ae1)
        {
            System.out.println("Unable to
            Close Socket.........!");
            System.exit(1);
        }
    }

    this.start();
}

public void run()
{
    String mradius;
    float radius;
    double area;
    try{

        //Write Message to Client
        toClient.println("Server
                        Ready.............!");

        //Read radius from client
        mradius = fromClient.readLine();
        radius = Float.parseFloat(mradius);
```

```
            System.out.println("Radius From
            Client : "+mradius);

            System.out.println("Processing Area
            !..............");

            area = (radius * radius * 3.14);
            System.out.println("Area : "+area);

            //Write Area to Client

            toClient.println(area);

            System.out.println("Writing Area to
            Socket");
        }

        catch(Exception ae2)
        {
            System.out.println("Unable to Process
            ....");
            System.exit(1);
        }

        finally
        {
            try{
                netClient.close();
            }
            catch(Exception ae3)
            {
                System.out.println("Unable to
                Close the Conneciton ");
            }
        }
}//end of run

}// end of Server Class

//----------------------------------------
```