**NETWORKING**

Networking is the process of connecting multiple local or remote computers together and transferring from one computer to another computer.

**How can data be transferred from one computer to another?**

It is possible when both machines follow a protocol.

**What is protocol?**

Protocol is a program. At the sender’s end, it represents high level data into its equivalent machine level data. At the receiving end, another protocol program receives this machine level data and represents in its equivalent high level data form.

**Classifications of protocol**

1. Connection oriented protocol programs
2. Connectionless protocol programs

**Connection oriented protocol programs**

The protocol program in which the path to send the data from one system to another system is pre – decided is called connection oriented protocol program.

It is based on two criteria:

1. Shortest path
2. Network traffic

After deciding the path, the data would be transferred in a continuous stream of bytes. This is the reason we make use of stream in connection oriented program to transfer the data.

**Advantage**

Data reaching to the destination is guaranteed.

**Dis – advantage**

Once the path is decided and the data is being transferred, the path cannot be changed in between. If there is traffic in the path, then it waits till the traffic is cleared and then data is reached.

So, it is guaranteed that data is reached to the destination, but when the data is reached is not specified.

**Transmission Control Protocol is connection oriented protocol.**

**Connection less protocol programs**

The protocol program in which the path to send the data from one system to another is not pre – determined is called connection less protocol program.

In this protocol program, after the data is being represented in its machine equivalent form will be broken or divided into groups is called packets. The data is then transferred in the form of packets. Each packet may follow its own path in the network to reach the destination.

Each packet consists

1. Data
2. Address of the destination
3. Information regarding hierarchy of the packets. (Metadata)

At the destination, each packet is buffered and once all the packets reached the destination; all the packets are re – grouped according to the metadata and then converted into the required format.

**Advantage**

Sudden burst of traffic will not affect the data transmission because packets have flexibility to take different paths in the network.

**Dis – advantage**

Data reaching to the destination is not guaranteed.

**User Datagram Protocol is connection less protocol.**

**Internet protocol** is a low level routing protocol, that breaks data into small packets and sends them across the network, which does not gurantee to deliver said packets to the destination.

**Internet Protocol Address** (IP Address) is an unique identification number of a computer on a network. It ranges from 0.0.0.0 to 255.255.255.255.

**Transmission Control Protocol** (TCP) enables two hosts to establish a connection and exchange streams of data. TCP guarantees the delivery of data and also guarantees that the packets will be delivered in the same order in which they were sent.

**User Datagram Protocol** (UDP) is a standard, low overhead, connectionless, host-to-host protocol that is used over the IP. It allows an application program on one computer to send a datagram to an application program of another computer.

**Request:** An input data sending via network to an application (server) that is running in a remote computer.

**Response:** An output coming from that application back to this client program.

**Client:** It is an application that allows sending request.

**Server:** It is software (set of programs) that receives request via network, process that request, generates response and sends it back to the client program.

**Hostname:** It is an alias name of the computer mapped with the IP address of the computer.

Default IP address of computer is 127.0.0.1 and default hostname is localhost.

**Hostfile path:** C:\Windows\System32\Drivers\etc\hostfile.

**Port Number:** It is an identification number of the server software. It is a 32-bit positive integer number, ranges from 0 to 65535. The port number 0 to 1024 is already registered for public servers, so we cannot release new server software with the port number within this range.

In a computer, N number of server can be installed, and all of them can be start at the same time if they have different port number.

We can install more than one server with the same port number, but we cannot start them all at a time. If we start more than one server with the same port number or if we restart the server while it is running, it will leads to an exception, java.net.BindException: Address already in use.

**URL:** Uniform Resource Locator, is an absolute path of the remote file in a server system.

**URI:** Uniform Resource Identifier, is a relative path of the remote file in a server system.

**Format of URL**

Protocol://hostname:port\_number/resourcepath/param1=value&param2=value...

**Server path:** Protocol://hostname:port

**URI:** resourcepath

**Query String:** param1=value&param2=value...

**For example**

http://192.168.1.14:8080/add/addition/fno=10&sno=20

**Explain the process of transferring the data?**

Every system will have buffer memory. The data which is supposed to be entering into the system or leave the system will pass through the buffer only.

Suppose, we have two machines **M1** and **M2** between which data is to be transferred.

Let **B1** is the buffer memory of **M1**, and **B2** is the buffer memory of **M2**.

Let **prog1.class** file is in **M1** and **prog2.class** file is in **M2**.

**M1** and **M2** are connected through the network.

Imagine **prog2.class** is supposed to receive the data.

1. Create the object of class in **B2** of **M2**.
2. Let call this object as **obj2**.
3. The reference of this object must be present in **prog2.class** file.
4. Let call this reference as **ref2**.
5. In order to connect to **M1**, **accept()** must be called from **prog2.class** file as **ref2.accept()**.
6. The basic functionality of **accept()** is to wait for the connection.
7. Since **prog1.class** file is not in execution. So, in order to start the execution, create the object of the class in **B1** of **M1**.
8. Let call this object as **obj1**.
9. The reference of this object must be present in **prog1.class** file.
10. Let call this reference as **ref1**.
11. **The important point is that obj2 must contain ref1 and obj1 must contain ref2.**
12. Once this process is finished, transmission starts.

Now assume, prog1.class file have to send some data to prog2.class file. Now prog2.class file have to receive the data, process the data and generate the result. The result has to be sent back to the prog1.class file.

To achieve the above goal, perform the following operations:

1. Using **ref1**, place the data from **prog1.class** file to obj1.
2. **obj1** contains **ref2**, which points to **obj2**. Using **ref2**, data is move from **obj1** to **obj2**.
3. Call **accept()** on **ref2**. With this, data present in **obj2** will be sent to **prog2.class**.
4. Process the data and generate the output.
5. Place the result to **obj2** using **ref2**.
6. **obj2** contains **ref1**, which points to **obj1**. Using **ref1**, result is move from **obj2** to **obj1**.
7. Using **ref1**, collect the result from **obj1** and transfer to **pro1.class**.

This is the process of transferring the data from one system to another.

**The objects which we are creating in the buffer to send/receive the data to/from the systems across the network are known as socket.**

**Socket:**Socket is a listener through which computers can receive requests and send responses. Using this listener, computer can connect to a network and communicate to another computer. It is the endpoint of the logical connection between two hosts.

**The transfer of data between the programs which are being executed in different systems is done through the socket. Hence, this is known as socket programming.**

The listener of client system is called **socket**.

The listener of server system is called **server socket**.

**How user can send request from client to server, when client system is connected with multiple servers?**

It is possible due to the URL format, which contains IP address/hostname of the server and resource path.

**How server can send response back to the same client when it is connected to the multiple clients?**

Due to virtual socket created in the server system for this socket. When a request is send from the client, in server system server socket creates a virtual socket for this client and stores the information about this client such as IP address/hostname, client application details... etc. Then after processing the request, server system sends back the response to the same client system using this virtual socket. (Actually the response is sent by virtual socket not by server socket).

**How can we perform network operations using Java applications?**

In **java.net** package, SUN has given required number of classes and interfaces to perform network operations.

Among the important classes are:

1. **Socket, Server Socket**

These two classes establish the communication between two computer using TCP/IP protocols.  
The following statement creates server socket.

ServerSocket <ref\_name> = new ServerSocket(port);

We can choose any port number currently not used by any other process.

After the server socket is created, the server can use the following statement to listen for connections:

Socket <ref\_name> = <ServerSocket ref\_name>.accept();

This statement waits until the client connects to the server socket.

The following statement requests the client to connect to the server.

Socket <ref\_name> = new Socket(serverName, port);

This statement opens a socket so that client program can communicate with the server. serverName is the server’s IP address.

Attempting to create a **ServerSocket** already in use throws the **java.net.BindExceptions**.

The **Socket** constructor throws **java.net.UnknownHostException** if the host not found.

1. **DatagramPacket, Datagram Socket**

These two classes establish the communication between two computers using UDP protocol.

**DatagramSocket** define four public constructors:

DatagramSocket() throws SocketException

DatagramSocket(int port\_number) throws SocketException

DatagramSocket(int port, InetAddress ipAddress) throws SocketException

DatagramSocket(SocketAddress address) throws SocketException

The first creates a DatagramSocket bound to any unused port on the local computer.

The second creates a DatagramSocket bound to the port specified by port\_number.

The third constructs a DatagramSocket bound to the specified port and InetAddress.

The fourth constructs a DatagramSocket bound to the specific SocketAddress.

SocketAddress is an abstract class that is implemented by the concrete class InetSocketAddress.

DatagramSocket defines two important methods. They are:

void send(DatagramPacket packet) throws IOException

void receive(DatagramPacket packet) throws IOException

The **send()** method sends the packet to the port specified by packet.

The **receive()** methods waits for the packet to be received from the port specified by packet.

**DatagramPacket** defines four public constructors:

DatagramPacket(byte data[], int size)

DatagramPacket(byte data[], int offset, int size)

DatagramPacket(byte data[], int size, InetAddress ipAddress, int port)

DatagramPacket(byte data[], int offset, int size, InetAddress ipAddress, int port)

The first constructor specifies the buffer that will receives data and size of the packet. It is used for receiving data over **DatagramSocket**.

The second constructor allows to creates offset in which data is stored.

The third constructor used the target address and the port, which is used by **DatagramSocket**to specify where the data in the packet will be sent.

The fourth constructor transmits packets at the beginning of the specified offset into the data.

DatagramPacket defines several packets, out of which two are most important:

InetAddress getAddress()

int getPort()

The **getAddress()** method returns the address of the source or destination.

The **getPort()** method returns the port number.

1. **URL, URLConnection**

These two classes are used to read and write data from/to a web resource (file available on server system).

1. **InetAddress**

It is used to retrieve IP address and Hostname of the computer dynamically.

**Program demonstrating networking using TCP protocols**

import java.net.Socket;

import java.io.\*;

class ClientSocket

{

public static void main(String[] args)

{

try

{

Socket s = new Socket("localhost",4446);

String data = "Data is sending from the client.....";

OutputStream os = s.getOutputStream();

PrintStream out = new PrintStream(os);

out.println(data);

}

catch (IOException ioe)

{

ioe.printStackTrace();

}

}

}

import java.net.ServerSocket;

import java.net.Socket;

import java.io.\*;

class ServerSocketDemo

{

public static void main(String[] args)

{

try

{

ServerSocket ss = new ServerSocket(4446);

Socket s = ss.accept();

InputStream is = s.getInputStream();

BufferedReader br = new BufferedReader(new InputStreamReader(is));

System.out.println(br.readLine());

}

catch (IOException ioe)

{

ioe.printStackTrace();

}

}

}

**Program to send radius to the server and get the area**

import java.net.\*;

import java.util.Scanner;

import java.io.\*;

class Client

{

public static void main(String[] args)

{

Socket s;

Scanner scn;

OutputStream os;

InputStream is;

DataOutputStream dos;

DataInputStream dis;

int radius;

double area;

try

{

s = new Socket("localhost", 4444);

scn = new Scanner(System.in);

os = s.getOutputStream();

dos = new DataOutputStream(os);

System.out.println("Enter the radius: ");

radius = scn.nextInt();

dos.writeInt(radius);

is = s.getInputStream();

dis = new DataInputStream(is);

area = dis.readDouble();

System.out.println("Area of the circle is: " + area);

dis.close();

is.close();

dos.close();

os.close();

s.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

**After this program, write below program to receive the radius and calculate the area of circle and returns value to the client.**

**Program to receive radius, calculate area of circle and return value to the client.**

import java.net.\*;

import java.io.\*;

class Server

{

public static void main(String[] args)

{

ServerSocket ss;

Socket s;

InputStream is;

OutputStream os;

DataInputStream dis;

DataOutputStream dos;

int radius;

double area;

try

{

ss = new ServerSocket(4444);

s = ss.accept();

is = s.getInputStream();

dis = new DataInputStream(is);

radius = dis.readInt();

area = 3.14F \* radius \* radius;

os = s.getOutputStream();

dos = new DataOutputStream(os);

dos.writeDouble(area);

dos.close();

os.close();

dis.close();

is.close();

s.close();

ss.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

**On – line chatting application**

import java.net.Socket;

import java.io.\*;

import java.util.Scanner;

class ChatClient

{

public static void main(String[] args)

{

Socket s;

OutputStream os;

InputStream is;

DataOutputStream dos;

DataInputStream dis;

String msgSend, msgReceived;

try

{

s = new Socket("localhost", 4444);

os = s.getOutputStream();

is = s.getInputStream();

dos = new DataOutputStream(os);

dis = new DataInputStream(is);

Scanner scn = new Scanner(System.in);

while (true)

{

msgSend = scn.nextLine();

dos.writeUTF(msgSend);

msgReceived = dis.readUTF();

System.out.println(msgReceived);

if (msgReceived == "bye")

break;

}

dis.close();

dos.close();

is.close();

os.close();

s.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

import java.net.\*;

import java.io.\*;

import java.util.Scanner;

class ChatServer

{

public static void main(String[] args)

{

ServerSocket ss;

Socket s;

InputStream is;

OutputStream os;

DataInputStream dis;

DataOutputStream dos;

String msgSend, msgReceived;

try

{

ss = new ServerSocket(4444);

s = ss.accept();

is = s.getInputStream();

os = s.getOutputStream();

dis = new DataInputStream(is);

dos = new DataOutputStream(os);

Scanner scn = new Scanner(System.in);

while (true)

{

msgReceived = dis.readUTF();

System.out.println(msgReceived);

if (msgReceived == "bye")

break;

msgSend = scn.nextLine();

dos.writeUTF(msgSend);

}

dos.close();

dis.close();

os.close();

is.close();

s.close(); ss.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

**Echo Server**

Client sends some data to the server and if the server sends the same data to the client, then that server is known as echo server.

**Note**

The **ChatServer** program in the above example is capable of serving only one request. To make it handle multiple request, put the logic inside an infinite loop (while(true)). Now the ChatServer program is capable of handling more than one request. But, this does not solve the problem completely, because the server is now handling multiple request, but not simultaneously.

In real time, we need to develop servers in such a way that they should be capable of handling multiple requests simultaneously. To achieve this goal, we need to make use of the concept of multi – threading.

**On – line chatting program using multi – threading**

import java.net.\*;

import java.io.\*;

import java.util.Scanner;

class ChatServer

{

public static void main(String[] args)

{

ServerSocket ss;

Socket s;

try

{

ss = new ServerSocket(4444);

while (true)

{

s = ss.accept();

ServerChat sc = new ServerChat(s);

sc.start();

}

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

import java.net.\*;

import java.io.\*;

import java.util.Scanner;

class ServerChat extends Thread

{

ServerSocket ss;

Socket s;

InputStream is;

OutputStream os;

DataInputStream dis;

DataOutputStream dos;

String msgSend, msgReceived;

ServerChat(Socket s)

{

this.s = s;

}

public void run()

{

try

{

is = s.getInputStream();

os = s.getOutputStream();

dis = new DataInputStream(is);

dos = new DataOutputStream(os);

Scanner scn = new Scanner(System.in);

while (true)

{

msgReceived = dis.readUTF();

System.out.println(msgReceived);

if (msgReceived == "bye")

break;

msgSend = scn.nextLine();

dos.writeUTF(msgSend);

}

dos.close();

dis.close();

os.close(); is.close();

s.close(); ss.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

import java.net.Socket;

import java.io.\*;

import java.util.Scanner;

class ChatClient

{

public static void main(String[] args)

{

Socket s;

OutputStream os;

InputStream is;

DataOutputStream dos;

DataInputStream dis;

String msgSend, msgReceived;

try

{

s = new Socket("localhost", 4444);

os = s.getOutputStream();

is = s.getInputStream();

dos = new DataOutputStream(os);

dis = new DataInputStream(is);

Scanner scn = new Scanner(System.in);

while (true)

{

msgSend = scn.nextLine();

dos.writeUTF(msgSend);

msgReceived = dis.readUTF();

System.out.println(msgReceived);

if (msgReceived == "bye")

break;

}

dis.close();

dos.close();

is.close();

os.close();

s.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

**Transferring data using connection – less protocol programming**

In this protocol programming, we need to create a socket (i.e. an object) which can transmit data in the form of packets. For this, we make use of **DatagramSocket** class.

To transmit the data in the form of packets, first of all the data should be represented in the form of packets and each packet follows a random path to reach the destination. This indicates that each packet should have the address of the destination in it.

So, the constructor of the **DatagramPacket**class is defined to accept four argumets:

**DatagramPacket(byte b[], int ba.length, InetAddress ia, int port\_no)**

The 1st argument represents that the data should be represented in the form of bytes.

The 2nd argument is the total length of the bytes (data).

The 3rd argument is the IP Address of the destination (server).

The 4th argument is the port number in which server program is running.

**Note:** In the connection oriented protocol programming, we had directly mentioned the address of the server system (i.e. localhost) while Socket object.

But in connectionless protocol programming we need to mention the IP Address in the form of an object of the **InetAddress** class.

**InetAddress ia = InetAddress.getByName(“HostName”);**

getByName() is a factory method, because it returns object of the class in which it is present.

We will make use of **send()** method of the **DatagramSocket** class to insert or release the packets into the network.

**// Program to demonstrate UDP connection**

import java.net.\*;

class UDPServer1

{

public static void main(String[] args) throws Exception

{

DatagramSocket ds = new DatagramSocket(6666);

byte ba1[] = new byte[100];

DatagramPacket dp1 = new DatagramPacket(ba1, 100);

ds.receive(dp1);

String data = new String(ba1);

System.out.println(data);

}

}

import java.net.\*;

class UDPClient1

{

public static void main(String[] args) throws Exception

{

String s1 = "Data sent from client";

DatagramSocket ds = new DatagramSocket(5555);

InetAddress ia = InetAddress.getByName("localhost");

byte ba1[] = s1.getBytes();

DatagramPacket dp1 = new DatagramPacket(ba1, ba1.length, ia, 6666);

ds.send(dp1);

}

}

**// Program to demonstrate sending and receiving message at a time**

import java.net.\*;

class UDPClient1

{

public static void main(String[] args) throws Exception

{

String s1 = "I am Sandeep Dev";

DatagramSocket ds = new DatagramSocket(5555);

InetAddress ia = InetAddress.getByName("localhost");

byte ba1[] = s1.getBytes();

DatagramPacket dp1 = new DatagramPacket(ba1, ba1.length, ia, 6666);

ds.send(dp1);

byte ba2[] = new byte[100];

DatagramPacket dp2 = new DatagramPacket(ba2, 100);

ds.receive(dp2);

String data = new String(ba2);

System.out.println(data);

}

}

import java.net.\*;

class UDPServer1

{

public static void main(String[] args) throws Exception

{

DatagramSocket ds = new DatagramSocket(6666);

byte ba1[] = new byte[100];

DatagramPacket dp1 = new DatagramPacket(ba1, 100);

ds.receive(dp1);

String data = new String(ba1);

System.out.println("Data received from the client" + data);

String s1 = "Data sent from Server";

byte ba2[] = s1.getBytes();

DatagramPacket dp2 = new DatagramPacket(ba2, ba2.length, dp1.getAddress(), dp1.getPort());

ds.send(dp2);

}

}

**How can the server know from which client it has received the data and how can it sent back the response to that particular client only?**

To receive the data, **receive()** method is being called. To receive() method, we pass the **DatagramPacket** classobject. This object holds the information of the client (This indicates that the packets along with holding the metadata and address and port number, also holds the details of the client). The client **IP Address** and **Port Number** of the client can be get by methods **getAddress()** and **getPort()** methods.

To send the data, **send()** method is being called on the object of the **DatagramSocket** class.The argument to this method is object of the **DatagramPacket** class.

This is how, the server recognizes the client after receiving the data from it, process the data and sends the processed data back to the client.

**How client will connect to the server when port number of the server program is not known?**

Whenever we do not know the port number of the server program, then for the client to get connected to the server, we make use of the URL (Universal Resource Locator) of the server program.

We have two classes – **URL** and **URLConnection** in the **java.net** package.

**URL** class is used to represent the URL in the form of an object.

**openConnection** is a non – static method of **URL** class which returns object of the **URLConnection** class.

Now by calling the functionalities present in the **URLConnection** class, on the object obtained in the above step, we can exactly establish the connection.

**Note**

Establishing the connection using the URL of the server program comes under connection – oriented protocol programming. So, here we need streams in order to send or receive data.

Before establishing connection we need to set some properties for the object of the **URLConnection** class. We need to mention whether we want to send or receive data using that object.

**A sample program in which client is connected to the server using the URL**

import java.io.\*;

import java.net.\*;

public class URLDemo

{

public static void main(String args[])

{

URL u;

URLConnection uc = null;

try

{

u = new URL("http://www.google.com");

try

{

uc = u.openConnection();

uc.setDoOutput(true);

uc.setDoInput(true);

uc.connect();

InputStream is = uc.getInputStream();

BufferedReader br = new BufferedReader(new InputStreamReader(is));

String data = br.readLine();

System.out.println(data);

}

catch (IOException ex)

{

ex.getMessage();

}

}

catch (MalformedURLException ex)

{

ex.getMessage();

}

}

}