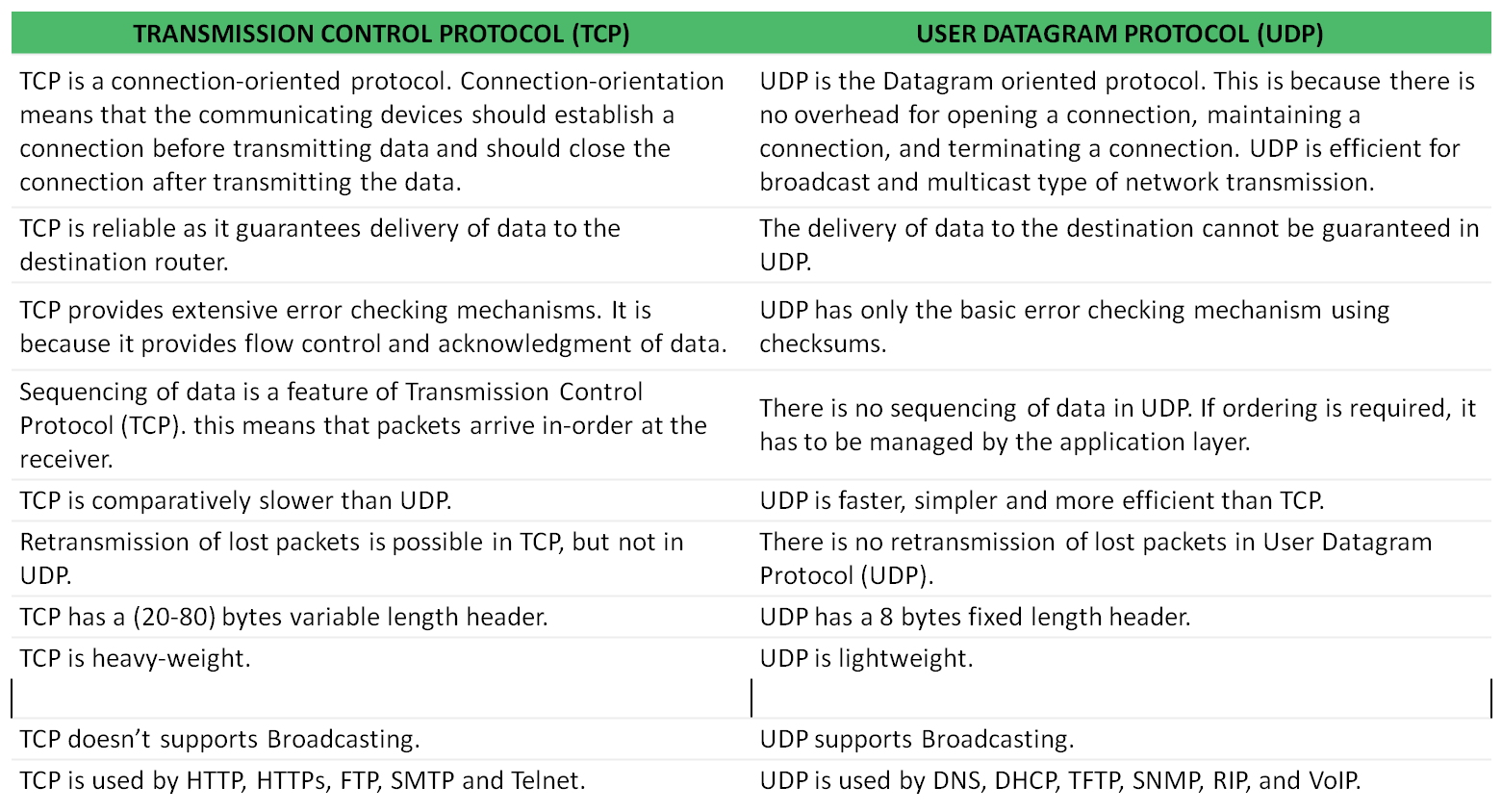
**Chapter 13 and Chapter 16**

**Client Server Communication**

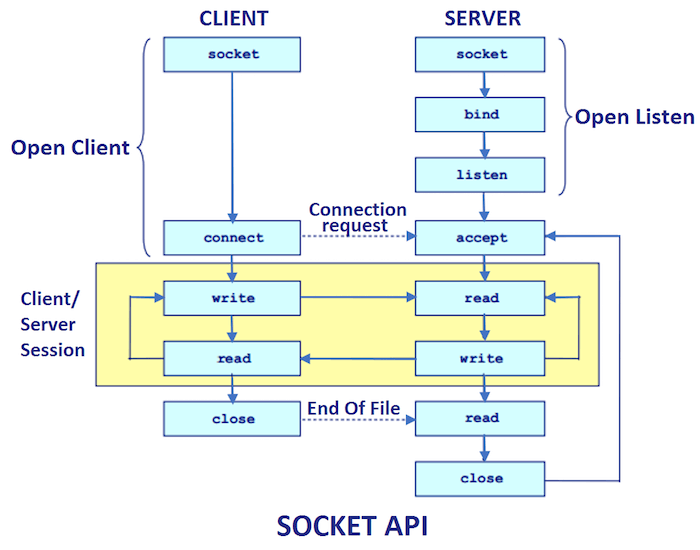
* Sockets allow communication between two different processes on the same or different machines.
* The Socket is an interface in which client and server communicate and pass the information to one another.
* Java Socket programming is used for communication between the applications running on different JRE.
* There are two communication protocols either one can use in socket programming:
  + - * Transfer Control Protocol (TCP)
      * User Datagram Protocol (UDP)
* TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.
* TCP is reliable as it guarantees delivery of data to the destination router.
* UDP is the conection-less protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection.
* The delivery of data to the destination cannot be guaranteed in UDP.
* This lecture presents a connection oriented socket programming.
* The two key classes from the java.net package used for connection-oriented socket programming
  + Socket
  + ServerSocket

***Difference between TCP and UDP***

****

* **The following steps occur when establishing a TCP connection between two computers using sockets**

1. The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
2. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.



**Steps for creating a simple server program in Java :**

1. Open the Server Socket:

ServerSocket Server = new ServerSocket(portNumber);

**Note:** Create Server Socket object by passing 1 arguments (i.e., portNumber).Use the same port number (i.e., "1342"), which has used in the client. For Example: ServerSocket S1 = new ServerSocket(1342);

1. Wait for the Client Request:

Socket client = server.accept();

**Note:** Listens for a connection to be made to this socket and accepts the request from client. The method blocks until a connection is made.

1. Create Input streams for receive from the client

Scanner SC = **new** Scanner(client.getInputStream());

**int** number=SC.nextInt();

or

DataInputStream is = new DataInputStream(client.getInputStream());

String line = is.readLine();

1. Create output streams for send to client:

PrintStream PS = **new** PrintStream(client.getOutputStream());

PS.println(“Hello”);

or

DataOutputStream os = new DataOutputStream(client.getOutputStream());

os.writeBytes(“Hello\n”);

1. Close socket:

client.close();

**Implementing a client consists of four basic steps:**

1. **Create** a Socket object.

Socket client = **new** Socket (hostName, portNumber);

**Note:** Create a Socket object by passing 2 arguments. One hostname (i.e., IP address of server machine), but here both client and server run in a same machine, hence use IP address"127.0.0.1" (i.e., localhost). Second, portNumber, use any number (i.e., "1342"). Same port number use in client and server. For Example: Socket S = **new** Socket("127.0.0.1", 1342);

1. **Create** an output stream that can be used to send information to the Socket.

PrintStream PC = **new** PrintStream(client.getOutputStream());

PC.println(number);

or

PrintWriter out = new PrintWriter(client.getOutputStream(), true);

out.println(number);

1. **Create** an input stream to read the response from the server.

Scanner SC1 = **new** Scanner(client.getInputStream());

**int** temp = SC1.nextInt();

or

BufferedReader in = new

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

1. Close the Socket when done.

client.close();

**client Socket Server**

**number=10 number number=10**

**result result result= number \* 2**

**System.out.println(result)**

**20**

**Example:** Write a client server program, where client accepts the number from the user and pass that number to the server. After that, server multiply that number by 2 and send back the resultant value to the client. Finally, the client print the resultant value on screen.

**Server side Program**

**import** java.io.IOException;

**import** java.io.PrintStream;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.util.Scanner;

**public** **class** Server {

**public** **static** **void** main(String[] args) **throws** IOException {

//Create Server Socket object by passing 1 arguments

//(i.e., Port number).Use the same port number (i.e., "1342"),

//which has used in the client.

ServerSocket S1 = **new** ServerSocket(1342);

//Listens for a connection to be made to this socket and accepts the

// request from client. The method blocks until a connection is made.

Socket SS = S1.accept();

//Accept the number by server, which sent by the client and

//store in number.

Scanner SC = **new** Scanner(SS.getInputStream());

**int** number=SC.nextInt();

//multiply the number by 2 and store in the result.

**int** result = 2 \* number;

//Now pass the result to the client. Hence first create the

// object of printStream and pass the result to client.

PrintStream PS = **new** PrintStream(SS.getOutputStream());

PS.println(result);

}

}

**Client Side Program**

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.PrintStream;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** Client {

**public** **static** **void** main(String aggs[]) **throws** UnknownHostException, IOException {

System.***out***.println("Enter any integer number");

//Create scanner object to take input from user

Scanner SC = **new** Scanner(System.***in***);

**int** number = SC.nextInt();

//Create Socket object by passing 2 arguments. One IP address // of server machine, but here both client and server run in

//a same machine, hence use IP address"127.0.0.1" (i.e., localhost). //Second, Port number, use any number (i.e., "1342"). same port //number use in client and server.

Socket S = **new** Socket("127.0.0.1", 1342);

//Now pass the number to the server. hence first create the object

//of printStream and pass the number to server.

PrintStream PC = **new** PrintStream(S.getOutputStream());

PC.println(number);

//use this socket object (i.e., S) to accept result from server and

//store in temp and display on client screen.

Scanner SC1 = **new** Scanner(S.getInputStream());

**int** temp = SC1.nextInt();

System.***out***.println("The Result is " + temp);

}

}

**Example:** Write a client server program, connection established sever sends a date object to client and finally, client print the date and time on screen.

**Server side Program**

import java.net.\*;

import java.util.Date;

import java.io.\*;

public class Server {

//\*\* The TCP port for the object time service. \*/

public static final short TIME\_PORT = 1951;

public static void main(String[] argv) {

ServerSocket sock;

Socket clientSock;

try {

//Create Server Socket object by passing 1 arguments

//(i.e., Port number). Use the same port number (i.e., "1951"),

//which has used in the client.

sock = new ServerSocket(TIME\_PORT);

// clientSock = sock.accept(), it listens for a connection to be

// made to this socket and accepts the request. The method

//blocks until a connection is made.

while ((clientSock = sock.accept()) != null) {

//clientSock.getInetAddress() method returns the remote

//IP address to which this socket is connected, or null

//if the socket is not connected.

System.out.println("Accept from "

+ clientSock.getInetAddress());

//Now pass the result to the client. hence first create the

//object of ObjectOutputStream and pass the Date and

//time to client.

ObjectOutputStream os =

new ObjectOutputStream(clientSock.getOutputStream());

//create a Date object and initializes it so that it represents

//the time at which it was allocated. write() method Write

//the specified object to the ObjectOutputStream.

os.writeObject(new Date());

os.close();

}

} catch (IOException e) {

System.err.println(e);

}

}

}

**Client Side Program**

**import** java.net.\*;

**import** java.util.Date;

**import** java.io.\*;

**public** **class** Client {

/\*\* The TCP port for the object time service. \*/

**public** **static** **final** **short** ***TIME\_PORT*** = 1951;

**public** **static** **final** String ***hostName*** = "localhost";

**public** **static** **void** main(String[] argv) {

**try** {

//Create Socket object by passing 2 arguments. One IP address

// of server machine, but here both client and server run in a // same machine, hence use IP address"127.0.0.1" (i.e.,

//localhost). Second, Port number, use any number (i.e.,

//"1951"). same port number use in client and server.

Socket sock = **new** Socket(***hostName***, ***TIME\_PORT***);

//Creates a BufferedInputStreamand saves its argument, the

// input stream in,for later use. An internalbuffer array is

//created and stored in buf.

BufferedInputStream bf = **new**

BufferedInputStream(sock.getInputStream());

//Creates an ObjectInputStream that reads from the specified

//InputStream.

ObjectInputStream is = **new** ObjectInputStream(bf);

//Read an object from the ObjectInputStream and validate the

//Object

Object o = is.readObject();

**if** (o == **null**) {

System.***err***.println("Read null from server!");

}

**else** **if** ((o **instanceof** Date)) {

// Valid, so cast to Date, and print

Date d = (Date) o;

System.***out***.println("Server host is " + ***hostName***);

System.***out***.println("Time there is " + d.toString());

} **else** {

**throw** **new** IllegalArgumentException("Wanted Date,

got " + o);

}

} **catch** (ClassNotFoundException e) {

System.***err***.println("Wanted date, got INVALID CLASS

(" + e + ")");

} **catch** (IOException e) {

System.***err***.println(e);

}

}

}

**Example:** Write a client server program where Server echoes clients’s msg.

**Server side Program**

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** EchoServer {

/\*\* Our server-side rendezvous socket \*/

**protected** ServerSocket sock;

/\*\* The port number to use by default \*/

**public** **final** **static** **int** ***ECHOPORT*** = 7;

/\*\* Flag to control debugging \*/

**protected** **boolean** debug = **true**;

/\*\* main: construct and run \*/

**public** **static** **void** main(String[] args) {

**int** p = ***ECHOPORT***;

**if** (args.length == 1) {

**try** {

p = Integer.*parseInt*(args[0]);

} **catch** (NumberFormatException e) {

System.***err***.println("Usage: EchoServer [port#]");

System.*exit*(1);

}

}

**new** EchoServer(p).handle();

}

/\*\* Construct an EchoServer on the given port number \*/

**public** EchoServer(**int** port) {

**try** {

sock = **new** ServerSocket(port);

} **catch** (IOException e) {

System.***err***.println("I/O error in setup");

System.***err***.println(e);

System.*exit*(1);

}

}

/\*\* This handles the connections \*/

**protected** **void** handle() {

Socket ios = **null**;

BufferedReader is = **null**;

PrintWriter os = **null**;

**while** (**true**) {

**try** {

System.***out***.println("Waiting for client...");

ios = sock.accept();

System.***err***.println("Accepted from " +

ios.getInetAddress().getHostName());

is = **new** BufferedReader(**new**

InputStreamReader(ios.getInputStream(), "8859\_1"));

os = **new** PrintWriter(**new**

OutputStreamWriter(ios.getOutputStream(), "8859\_1"), **true**);

String echoLine;

**while** ((echoLine = is.readLine()) != **null**) {

System.***err***.println("Read " + echoLine);

os.print(echoLine + "\r\n");

os.flush();

System.***err***.println("Wrote " + echoLine);

}

System.***err***.println("All done!");

}**catch** (IOException e) {

System.***err***.println(e);

} **finally** {

**try** {

**if** (is != **null**)

is.close();

**if** (os != **null**)

os.close();

**if** (ios != **null**)

ios.close();

} **catch** (IOException e) {

// These are unlikely, but might indicate that

// the other end shut down early, a disk filled up

// but wasn't detected until close, etc.

System.***err***.println("IO Error in close");

}

}

}

}

}

**Client Side Program**

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.PrintWriter;

**import** java.net.Socket;

**public** **class** EchoClientOneLine {

/\*\* What we send across the net \*/

String mesg = "Hello across the net";

**public** **static** **void** main(String[] argv) {

**if** (argv.length == 0)

**new** EchoClientOneLine().converse("localhost");

**else**

**new** EchoClientOneLine().converse(argv[0]);

}

/\*\* Hold one conversation across the net \*/

**protected** **void** converse(String hostName) {

**try** {

Socket sock = **new** Socket(hostName, 7); // echo server.

BufferedReader is = **new** BufferedReader(**new**

InputStreamReader(sock.getInputStream()));

PrintWriter os = **new** PrintWriter(sock.getOutputStream(), **true**);

// Do the CRLF ourself since println appends only a \r on

// platforms where that is the native line ending.

os.print(mesg + "\r\n"); os.flush();

String reply = is.readLine();

System.***out***.println("Sent \"" + mesg + "\"");

System.***out***.println("Got \"" + reply + "\"");

} **catch** (IOException e) {

System.***err***.println(e);

}

}

}