```
class Client1
{
        public static void main(String args[ ])
        throws Exception
        {
                //create client socket with same port number
                Socket s = new Socket("localhost", 777);
                //to read data coming from server, attach InputStream to the socket
                InputStream obj = s.getInputStream();
                //to read data from the socket into the client, use BufferedReader
                BufferedReader br = new BufferedReader(new InputStreamReader(obj));
                //receive strings
                String str;
                while((str = br.readLine()) != null)
                System.out.println("From server: "+str);
                //close connection by closing the streams and sockets
                br.close();
                s.close();
        }
}
```

Output:

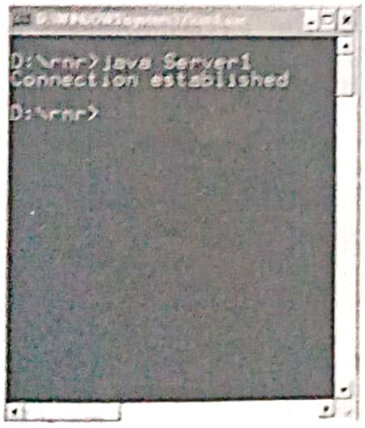
```
D:\rnr> javac Client1.java
D:\rnr>
```

To run the server and client in the same system:

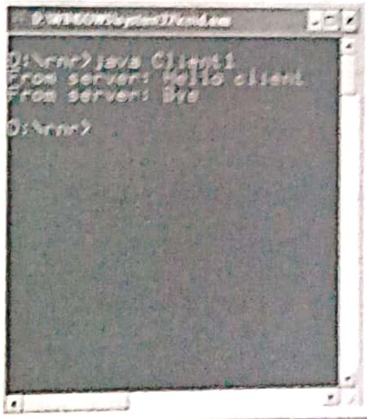
After compiling server1.java and client1.java, run these programs in two separate dos windows, as shown in the output.

To run in different systems:

Run the server1.java in a computer and client1.java in another. They should have been connected in a network.



```
D:\rnr>java Server1
Connection established

D:\rnr>
```



```
D:\rnr>java Client1
From server: Hello client
From server: Bye

D:\rnr>
```

Run the `Server1.java` in a DOS window, the server would be in waiting state, expecting a connection from a client. Then run `Client1.java` in another DOS window. Immediately the connection is established, and at server side it displays "Connection established". Then it sends two strings "Hello client" and "Bye" to the client, which are received and displayed at client terminal. When the server disconnects, it sends a null string to the client. When client receives null, it also disconnects.

Two-way Communication between Server and Client

It is possible to send data from the server and receive the response from the client. Similarly, the client can also send and receive the data to-and-fro. For this purpose, we need additional streams both at server and client. For example, to receive data into the server, it is a better idea to use BufferedReader as:

```
InputStream obj = s.getInputStream();
BufferedReader br = new BufferedReader(new InputStreamReader(obj));
```

Then read() or readLine() methods of BufferedReader class can be used to read data.

To send data from the client, we can take the help of DataOutputStream as:

```
OutputStream obj = s.getOutputStream();
DataOutputStream dos = new DataOutputStream(obj);
```

Then writeBytes() method of DataOutputStream can be used to send strings in the form of group of bytes.

Program 6: Write a program to create a server such that the server receives data from the client using BufferedReader and then sends reply to the client using PrintStream.

```
//Server2 - A server that receives data and sends data
import java.io.*;
import java.net.*;
class Server2
{
        public static void main(String args[ ])
        throws Exception
        {
                //Create server socket
                ServerSocket ss = new ServerSocket(888);
                //connect it to client socket
                Socket s = ss.accept();
                System.out.println("Connection established");
                //to send data to the client
                PrintStream ps = new PrintStream(s.getOutputStream());
                //to read data coming from the client
                BufferedReader br = new BufferedReader(new
                 InputStreamReader(s.getInputStream()));
                //to read data from the key board
                BufferedReader kb = new BufferedReader(new
                 InputStreamReader(System.in));
                while(true)  //server executes continuously
                {
                        String str,str1;
                        //repeat as long as client does not send null string
                        while((str = br.readLine()) != null)  //read from client
                        {
                                System.out.println(str);
                                str1 = kb.readLine();
                                ps.println(str1); //send to client
                        }
                        //close connection
                        ps.close();
                        br.close();
                        kb.close();
                        ss.close();
                        s.close();
                        System.exit(0); //terminate application
                } //end of while
        }
}
```

Output:

```
D:\rnr> javac Sever2.java
```

```
D:\rnr>
```

Program 7: Write a program to create a client which first connects to a server, then starts the communication by sending a string to the server. The server sends response to the client. When 'exit' is typed at client side, the program terminates.

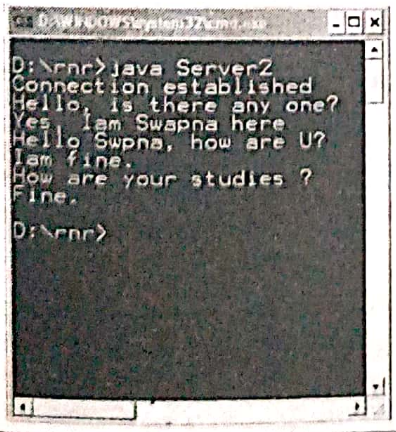
```
//Client2 - a client that sends data and receives also
import java.io.*;
import java.net.*;
class Client2
{
        public static void main(String args[ ])
        throws Exception
        {
                //Create client socket
                Socket s = new Socket("localhost", 888);
                //to send data to the server
                DataOutputStream dos = new DataOutputStream(s.getOutputStream());
                //to read data coming from the server
                BufferedReader  br = new BufferedReader(new
                 InputStreamReader(s.getInputStream()));
                //to read data from the key board
                BufferedReader kb = new BufferedReader(new
                 InputStreamReader(System.in));
                String str,str1;
                //repeat as long as exit is not typed at client
                while(!(str = kb.readLine()).equals("exit"))
                {
                        dos.writeBytes(str+"\n");  //send to server
                        str1 = br.readLine(); //receive from server
                        System.out.println(str1);
                }
                //close connection.
                dos.close();
                br.close();
                kb.close();
                s.close();
        }
}
```

Output:

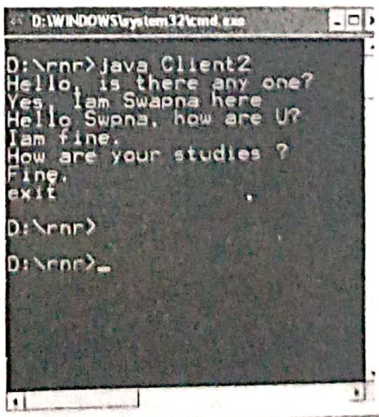
```
D:\rnr> javac Client2.java
D:\rnr>
```

Run the `server2.java` and `client2.java` in two dos windows.

See the output while running these programs.



```
D:\rnr>java Server2
Connection established
Hello, is there any one?
Yes, Iam Swapna here
Hello Swpna, how are U?
Iam fine.
How are your studies ?
Fine.

D:\rnr>
```



```
D:\rnr>java Client2
Hello, is there any one?
Yes, Iam Swapna here
Hello Swpna, how are U?
Iam fine.
How are your studies ?
Fine.
exit

D:\rnr>

D:\rnr>_
```

Retrieving a file at server

Let us write client and server programs, such that the client sends the name of a file to the server. After receiving the filename, the server searches for the file to know if it exists or not. If the file exists, the server sends the file contents to the client. This is shown in Programs 8 and 9, which are self-explanatory.

Program 8: Write a program that accepts the filename and checks for its existence. When the file exists at server side, it sends its contents to the client.

```
//A server that sends a file content to the client
import java.io.*;
import java.net.*;
class FileServer
{
    public static void main(String args[ ]) throws Exception
    {
        //create server socket
        ServerSocket ss = new ServerSocket(8888);

        //make the server wait till a client accepts connection
        Socket s = ss.accept();
        System.out.println("Connection established");

        //to accept file name from client
        BufferedReader in = new BufferedReader(new
         InputStreamReader(s.getInputStream()));

        //to send file contents to client
        DataOutputStream out = new DataOutputStream(s.getOutputStream());

        //read the filename from the client
        String fname = in.readLine();

        FileReader fr = null;
        BufferedReader file = null;
        boolean flag;

        //create File class object with filename
        File f = new File(fname);

        //test if file exists or not
        if(f.exists()) flag = true;
        else flag = false;

        //if file exists, send "Yes" to client, else send "No"
        if(flag == true) out.writeBytes("Yes"+"\n");
        else out.writeBytes("No"+"\n");

        if(flag == true)
        {
                //attach file to the FileReader to read data
                fr = new FileReader(fname);

                //attach FileReader to BufferedReader
                file = new BufferedReader(fr);

                String str;

                //read from BufferedReader and write to DataOutputStream
                while((str = file.readLine()) != null)
                {
                        out.writeBytes(str+"\n");
                }
                out.close();
                in.close();
                fr.close();
                s.close();
                ss.close();
        }
    }
}
```

utput:

```
D:\rnr> javac FileServer.java
D:\rnr>
```

his is a server program that receives the file name from the client and if file exists, it sends "Yes", therwise "No" to the client. This helps the client to understand whether the file really exists at erver or not. Then this server program sends the file contents to the client if the file exists.

rogram 9: Write a client program to accept a file name from the keyboard and send that name to he server. The client receives the file contents from the server.

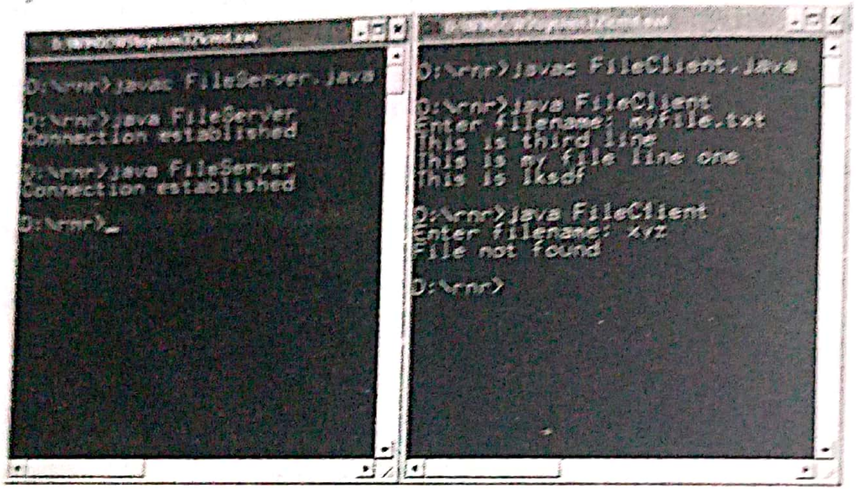
```
//FileClient - receiving  a file content
import java.io.*;
import java.net.*;

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

        //Create client socket
        Socket s = new Socket("localhost", 8888);
        //accept filename from keyboard
        BufferedReader kb = new BufferedReader(new
         InputStreamReader(System.in));

        System.out.print("Enter filename: ");
        String fname = kb.readLine();

        //send filename to the server using DataOutputStream
        DataOutputStream out = new DataOutputStream(s.getOutputStream());
        out.writeBytes(fname+"\n");

        //to read data coming from the server
        BufferedReader  in = new BufferedReader(new
         InputStreamReader(s.getInputStream()));

        String str;

        //read first line from server into str
        str = in.readLine();

        //if file is found server returns "Yes", else "No"
        if(str.equals("Yes"))
        {
                //read and display the file contents coming from server
                while((str = in.readLine()) != null)
                System.out.println(str);

                //close connection by closing the streams.
                kb.close();
                out.close();
                in.close();
                }
                else System.out.println("File not found");

        }
    }
```

Output:

```
    D:\rnr> javac FileClient.java
    D:\rnr>
```

This is a client program that accepts a file name from the keyboard and sends that name to the server. Then it reads the first line sent by the server. If it is *Yes*, then the file exists at server. If it is *No* then the file is not found at server. If file exists, then its contents are displayed at the client.

Run the fileserver.java and fileclient.java programs in two dos windows.



```
D:\rnr>javac FileServer.java

D:\rnr>java FileServer
Connection established

D:\rnr>java FileServer
Connection established

D:\rnr>_
```

```
D:\rnr>javac FileClient.java

D:\rnr>java FileClient
Enter filename: myfile.txt
This is third line
This is my file line one
This is lksdf

D:\rnr>java FileClient
Enter filename: xyz
File not found

D:\rnr>
```

Conclusion

The classes of `java.net` package internally use TCP/IP and UDP protocols that are responsible for sending and receiving data. We can also establish communication between a server and a client by creating server socket and client socket. This is called 'socket programming'. The data can be then sent or received between sockets, with the help of streams. However, socket programming offers only basic networking. If we want to achieve sophisticated client-server communication, we should look forward to servlets, JSPs (Java Server Pages), etc.