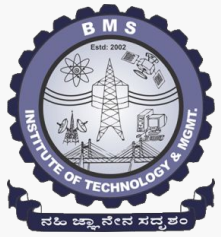


BMSIT&M – Dept. Of I.S.E

‘TRANSFROM HERE’

CN LAB - 15CSL78

SOCKET PROGRAM-TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol)

- **TCP/IP** specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination.
- **TCP** defines how applications can create channels of communication across a network. It also manages how a message is assembled into smaller packets before they are then transmitted over the internet and reassembled in the right order at the destination
- **IP** defines how to address and route each packet to make sure it reaches the right destination. Each gateway computer on the network checks this IP address to determine where to forward the message.

SOCKETS

- What are Sockets?
  Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors.

- Where Sockets are used?
  A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.
- Types of Socket:
  There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

- **Stream Sockets** – Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order – "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator
- **Datagram Sockets** – Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets – you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
- **Raw Sockets** – These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol.
- **Sequenced Packet Sockets** – They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications.

AIM:

Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Basic Concept : A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

Source Code:

TCP Client

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.Scanner;
class Client
{
        public static void main(String args[])throws Exception
    {
        String address = "";
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Server Address: ");
        address=sc.nextLine();
            //create the socket on port 5000
        Socket s=new Socket(address,5000);
        DataInputStream din=new DataInputStream(s.getInputStream()); DataOutputStream
        dout=new DataOutputStream(s.getOutputStream()); BufferedReader br=new
        BufferedReader(new InputStreamReader(System.in));
        System.out.println("Send Get to start...");
        String str="",filename="";
        try
        {
            while(!str.equals("start"))
            str=br.readLine();
            dout.writeUTF(str);
            dout.flush();
            filename=din.readUTF();
            System.out.println("Receving file: "+filename);
            filename="client"+filename;
    System.out.println("Saving as file: "+filename);


    long sz=Long.parseLong(din.readUTF()); System.out.println ("File Size:
    "+(sz/(1024*1024))+" MB");
    byte b[]=new byte [1024];
    System.out.println("Receving file..");
        FileOutputStream fos=new FileOutputStream(new File(filename),true);
        long bytesRead;
        do
        {
                bytesRead = din.read(b, 0, b.length);
                fos.write(b,0,b.length);
         }while(!(bytesRead<1024));
                System.out.println("Comleted");
                fos.close();
                dout.close();
                s.close();
        }
        catch(EOFException e)
        {
           //do nothing
        }
     }
}
```

TCP Server

```
    import java.io.DataInputStream;
    import java.io.DataOutputStream;
    import java.io.File;
    import java.io.FileInputStream;
    import java.net.ServerSocket;
    import java.net.Socket;
    import java.util.Scanner;
    class Server
    {
            public static void main(String args[])throws Exception
            {
               String filename;
               System.out.println("Enter File Name: ");
               Scanner sc=new Scanner(System.in);
               filename=sc.nextLine();
               sc.close();
               while(true)
               {
                                    //create server socket on port 5000
                        ServerSocket ss=new ServerSocket(5000);
                        System.out.println ("Waiting for request");
                        Socket s=ss.accept();
                        System.out.println ("Connected With "+s.getInetAddress().toString());
                        DataInputStream din=new DataInputStream(s.getInputStream());
                        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
```

```
if(!str.equals("stop")){
System.out.println("Sending File: "+filename);
dout.writeUTF(filename);
dout.flush();
File f=new File(filename);
FileInputStream fin=new FileInputStream(f);
long sz=(int) f.length();
byte b[]=new byte [1024];
int read;
dout.writeUTF(Long.toString(sz));
dout.flush();
System.out.println ("Size: "+sz);

                                System.out.println ("Buf size: "+ss.getReceiveBufferSize());
                                while((read = fin.read(b)) != -1) {
                                        dout.write(b, 0, read);
                                        dout.flush();
                                }
                                fin.close();
                                System.out.println("..ok");
                                dout.flush();
                        }
                        dout.writeUTF("stop");
                        System.out.println("Send Complete");
                        dout.flush();
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                        System.out.println("An error occured");
                }
                din.close();
                s.close();
                ss.close();
        }
    }
}
```

Few Classes & Functions Used:

- The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests.
- **Public ServerSocket(int port) throws IOException:** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
- The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other.
- The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.
- The **Java.io.FileInputStream** class obtains input bytes from a file in a file system.

- **public void bind(SocketAddress host, int backlog)**

Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor.

- **public InputStream getInputStream() throws IOException**

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

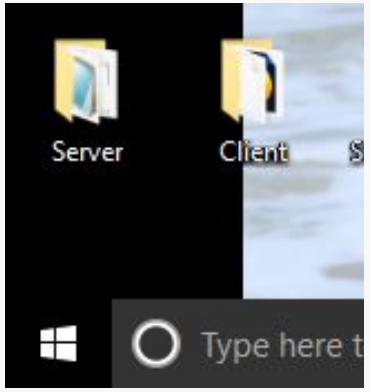
- **public OutputStream getOutputStream() throws IOException**

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

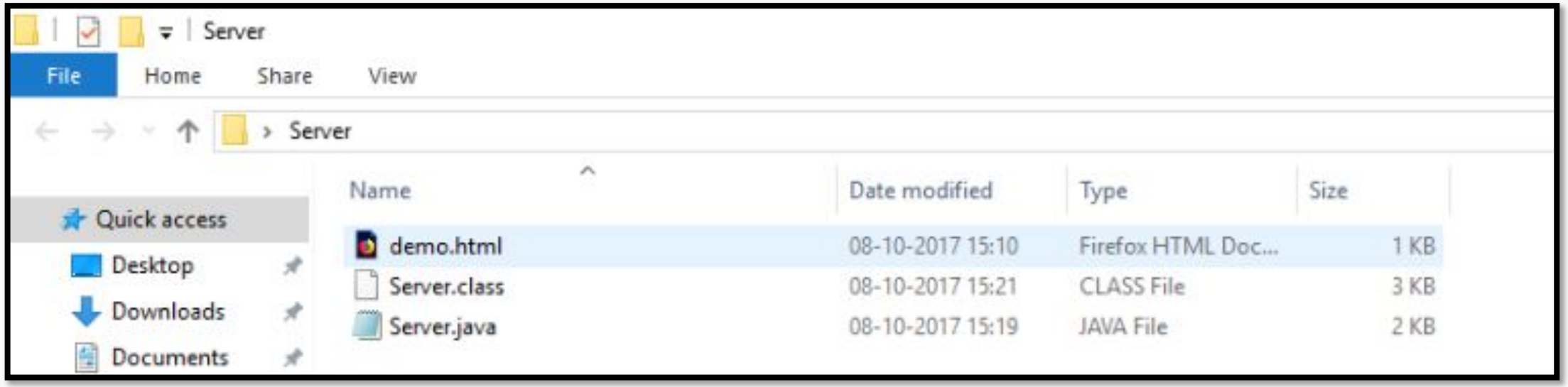
- The **java.io.DataOuputStream.writeUTF(String str)** method writes a string to the underlying output stream using modified UTF-8 encoding.

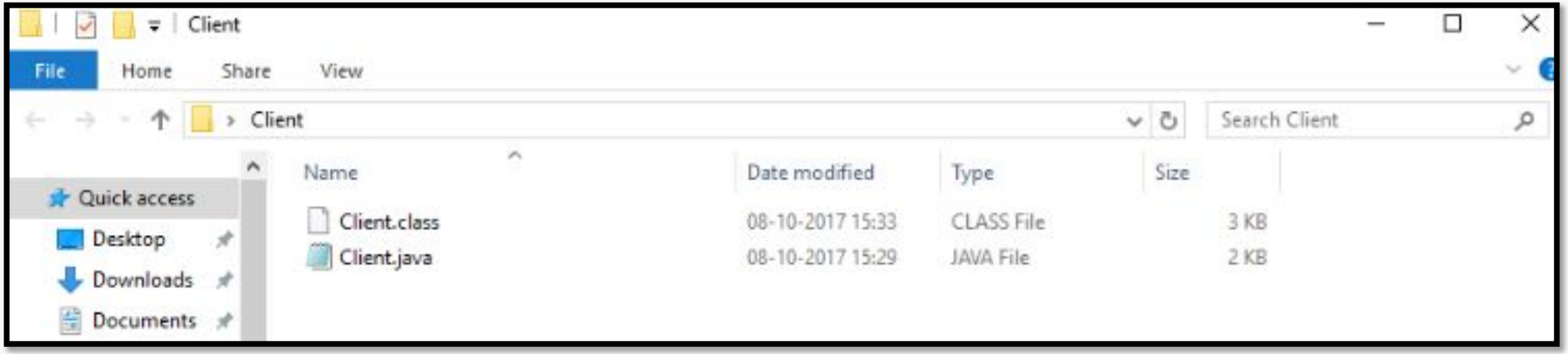
Execution & Output :

- Create two folders ,Server &Client on desktop.
- Paste the server.java and client.java files in respective folders.
- Also store the file you want to transfer in the Server folder.
- Open 2 terminal of command prompt ,one for each.
- Compile and run the java files.
- Give the server address as 127.0.0.1.

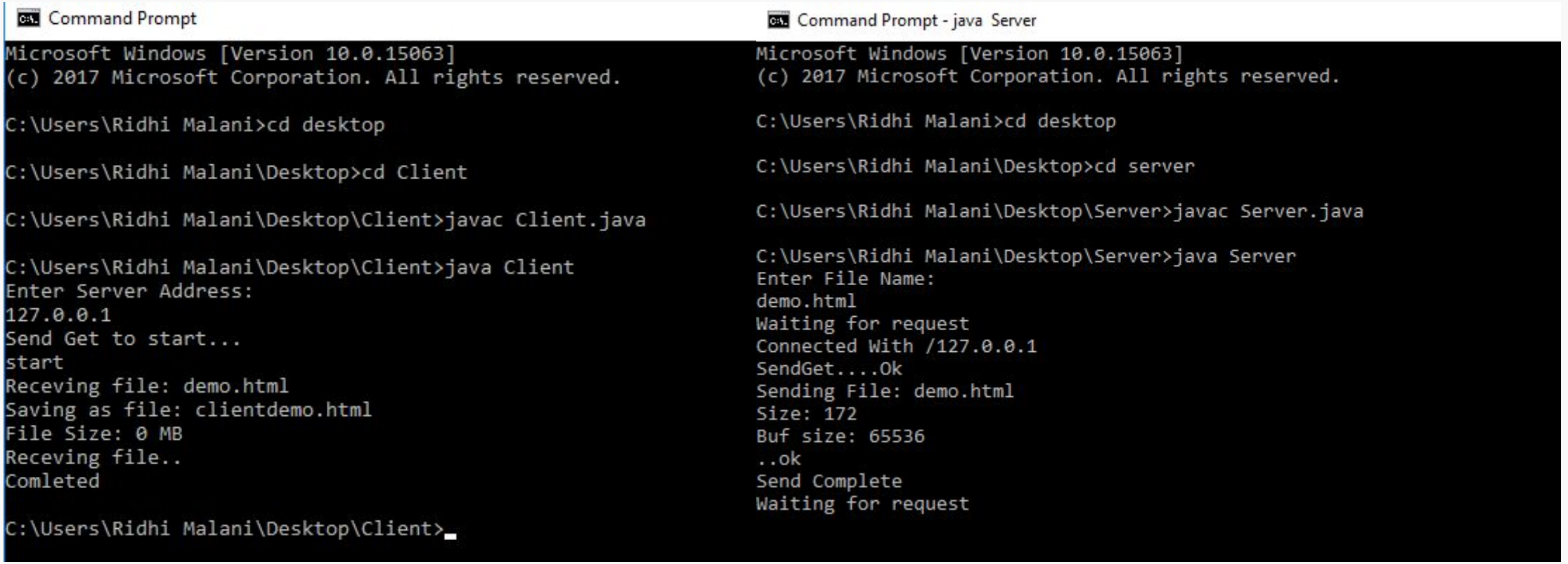


After Compilation of '.java' files :

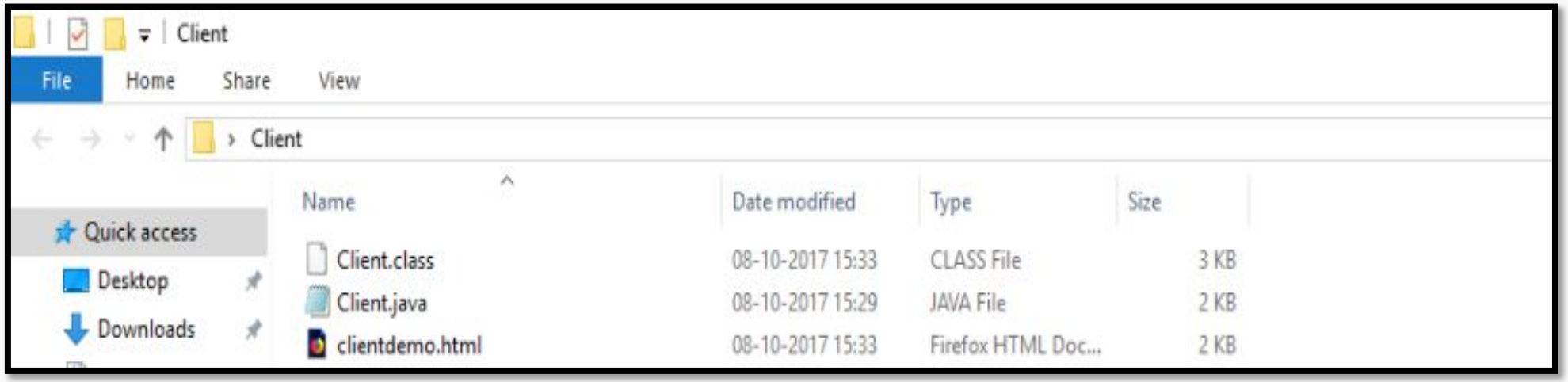




EXECUTION:


Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Users\Ridhi Malani>cd desktop
C:\Users\Ridhi Malani\Desktop>cd Client
C:\Users\Ridhi Malani\Desktop\Client>javac Client.java
C:\Users\Ridhi Malani\Desktop\Client>java Client
Enter Server Address:
127.0.0.1
Send Get to start...
start
Receving file: demo.html
Saving as file: clientdemo.html
File Size: 0 MB
Receving file..
Comleted
C:\Users\Ridhi Malani\Desktop\Client>
Command Prompt - java Server
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Users\Ridhi Malani>cd desktop
C:\Users\Ridhi Malani\Desktop>cd server
C:\Users\Ridhi Malani\Desktop\Server>javac Server.java
C:\Users\Ridhi Malani\Desktop\Server>java Server
Enter File Name:
demo.html
Waiting for request
Connected With /127.0.0.1
SendGet....Ok
Sending File: demo.html
Size: 172
Buf size: 65536
..ok
Send Complete
Waiting for request

OUTPUT:


Client
File
Home
Share
View
Client
Quick access
Desktop
Downloads
Name
Date modified
Type
Size
Client.class
08-10-2017 15:33
CLASS File
3 KB
Client.java
08-10-2017 15:29
JAVA File
2 KB
clientdemo.html
08-10-2017 15:33
Firefox HTML Doc...
2 KB

Submitted By:

RIDHI

MALANI

IBY15150045

THANK YOU.