Ex. No: 1

DATE: **Find the Web site address using Socket programming**

AIM:

To find the website address using socket programming

ALGORITHM:

Step 1: Start the process.

Step 2: Initialize the file stream and find the size of file using available()method.

Step 3: Read a character from the file and print it using Print stream object

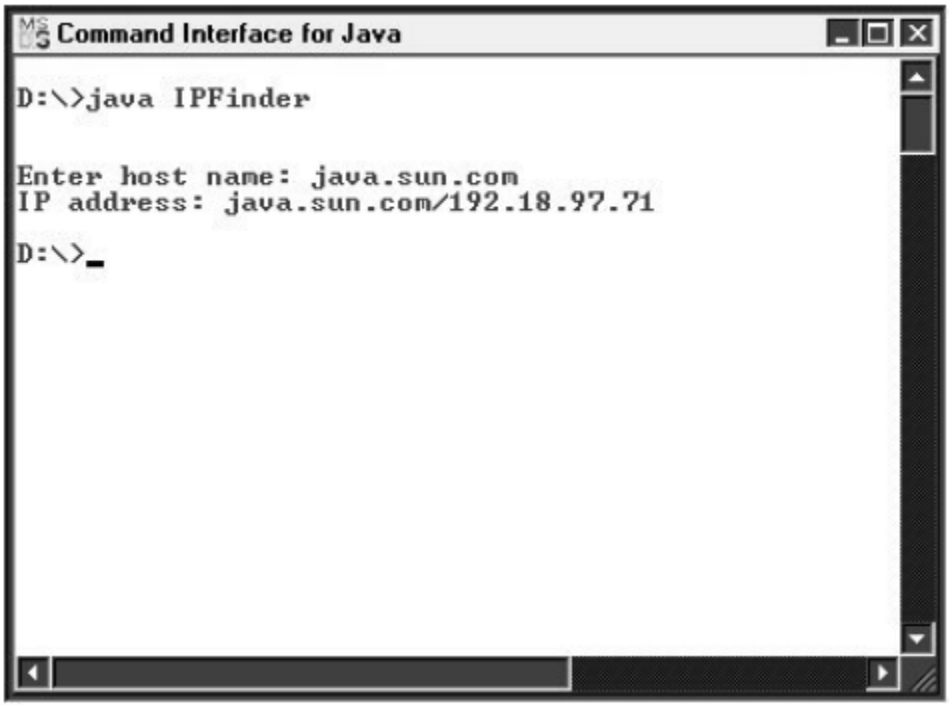
Step 4: using Inet address to find the host ip address

Step 5: Close print stream

Step 6: Stop the process

```
import java.net.*;
import java.util.*;
public class IPFinder
{
public static void main(String[] args)
{
String host;
Scanner input = new Scanner(System.in);
System.out.print("\n\nEnter host name: ");
host = input.next();
try
{
InetAddress address = InetAddress.getByName(host);
System.out.println("IP address: " + address.toString());
}
catch (UnknownHostException uhEx)
{
System.out.println("Could not find " + host);
}
}
}
```

Out put


Command Interface for Java
D:\>java IPFinder
Enter host name: java.sun.com
IP address: java.sun.com/192.18.97.71
D:\>

Ex. No: 2

DATE: **Find the local host IP address using Socket programming**

AIM:

To find the local host IP address using socket programming

ALGORITHM:

Step 1: Start the process.

Step 2: Initialize the file stream and find the size of file using available()method.

Step 3: Read a character from the file and print it using Print stream object

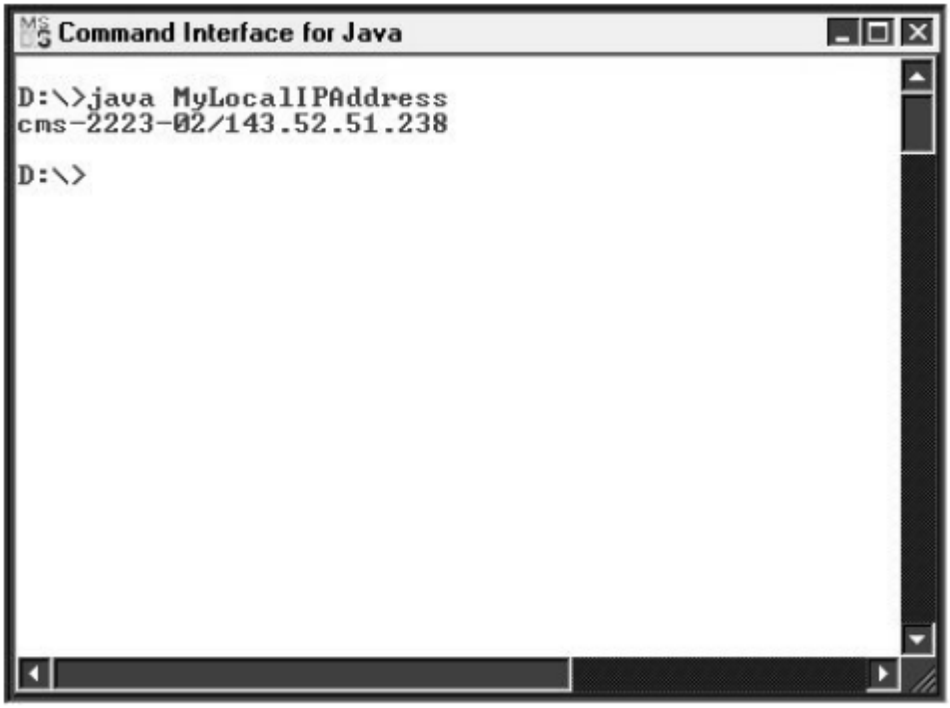
Step 4: using Inet address to find the host ip address

Step 5: Close print stream

Step 6: Stop the process

```
import java.net.*;
public class MyLocalIPAddress
{
public static void main(String[] args)
{
try
{
InetAddress address =InetAddress.getLocalHost();
System.out.println(address);
}
catch (UnknownHostException uhEx)
{
System.out.println("Could not find local address!");
}
}
}
```

Output


Command Interface for Java
D:\>java MyLocalIPAddress
cms-2223-02/143.52.51.238
D:\>

Ex. No: 3

DATE: SOCKET PROGRAMMING USING UDP

AIM:

To transfer a file from the client to server using TCP/IP Protocol.

ALGORITHM:

CLIENT SIDE

Step 1: Start the process.

Step 2: Initialize the file stream and find the size of file using available()method.

Step 3: Initialize the socket with the Port ID

Step 4: Read a character from the file and print it using Print stream object

Step 5: Close print stream

Step 6: Stop the process

SERVER SIDE

Step 1: Start the process.

Step 2: Import the net package. Initialize a file pointer and buffered writer object.

Step 3: Initialize the server socket with the Port ID as given in the client

Step 4: Initialize the server socket to accept the data send by the clients.

Step 5: Initialize the buffered reader object

Step 6: Write the data received in to the file until null character is reached

Step 7: Close the File and Socket Objects

Step 8: Stop the process

CLIENT PROGRAM:

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

class DatagramClient
{
	public static DatagramSocket ds;
	public static byte buffer[]=new byte[1024];
	public static int clientport=789, serverport=790;

	public static void main(String args[]) throws Exception
	{
		ds=new DatagramSocket(clientport);
		System.out.println("Client is waiting for servder to send data");
		System.out.println("Press Ctrl+C to come out");

		while(true)
		{
			DatagramPacket dp=new DatagramPacket(buffer, buffer.length);
			ds.receive(dp);
			String pdata=new String(dp.getData(),0, dp.getLength());
			System.out.println(pdata);
		}
	}
}
```

SERVER PROGRAM:

```
import java.net.*;

import java.io.*;

class DatagramServer

{

        public static DatagramSocket ds;

        public static int clientport=789, serverport=790;


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

        {

                byte buffer[]=new byte[1024];


                ds=new DatagramSocket(serverport);

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

                System.out.println("Server is waiting for input");

                while(true)

                {       String str=breader.readLine();

                        if(str==null || str.equals("End"))

                                break;

                        buffer=str.getBytes();

                        ds.send(new DatagramPacket(buffer, str.length(),
                        InetAddress.getLocalHost(),clientport));

                }

        }

}
```

OUTPUT:

SERVER SIDE:

```
Command Prompt - java DatagramClient
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Siva>d:

D:\>set path=C:\Program Files\Java\jdk1.6.0_16\bin

D:\>cd tcp1

D:\tcp1>javac DatagramClient.java

D:\tcp1>java DatagramClient
Client is waiting for servder to send data
Press Ctrl+C to come out
ibri college of technology
```

CLIENT SIDE:

```
Command Prompt - java DatagramServer
D:\tcp1>javac DatagramServer.java

D:\tcp1>java DatagramServer
Server is waiting for input
ibri college of technology
```

RESULT:

Thus the socket programming using UDP has been executed successfully.

EX. NO: 4

DATE: **SOCKET PROGRAMMING USING TCP**

AIM:

To write a program to transfer string between client and server using TCP

ALGORITHM:

Step 1: Start the program

Step 2: Include the import java.net.*;

Import java.lang.*;Package

Step 3: Define a class for implementing main method.

Step 4: Port is the number that indicates what kind of protocol a server on the Internet is using.

Step 5: A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and noted.

Step 6: Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet.

Step 7: The string is transferred by client to server. If the server is ready it sends the sequence of the requested string to the client.

Step 8: Print out with the necessary details.

SERVER:

```
import java.io.*;

import java.net.*;

class tcpserver1

{

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

 {

  ServerSocket ss=new ServerSocket(55);

  Socket s=ss.accept();

  DataInputStream in=new DataInputStream(s.getInputStream());

  DataOutputStream out=new DataOutputStream(s.getOutputStream());

  DataInputStream sysin=new DataInputStream(System.in);

  System.out.println("Enter an string:");

  String str=sysin.readLine();

  out.writeBytes(str+"\n");

  System.out.println("The string from TcP server"+in.readLine());

  }

}
```

CLIENT

```
import java.net.*;

import java.io.*;

public class tCPclient1

{

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

 {

 Socket s=new Socket("localhost",55);

 DataInputStream in=new DataInputStream(s.getInputStream());

 DataOutputStream out=new DataOutputStream(s.getOutputStream());

 String str=in.readLine();

 System.out.println("string from TCP server"+str);

 }

}
```

OUTPUT:

D:\java\MEPRACTICE>java server

CRICKET WORLD CUP

D:\java\MEPRACTICE>java client

CRICKET WORLD CUP

RESULT:

Thus the socket programming using TCP has been executed successfully.

EX. NO: 5

DATE: IMPLEMENTATION OF PING

AIM:

To implement ping command to determine the connectivity of a host in a network.

ALGORITHM:

Step 1: Create a object for runtime, process classes.

Step 2: Create an object 'r' for runtime.

Step 3: Get the runtime environment by calling getRuntime() method and assign it to r.

Step 4: Execute the ping command using the exec() method of Runtime class.

Step 5: Store the result in 'p', object of process class, if object value is null

Step 6: Create a string object to get IP address.

Step 7: If it starts with request, the n print "there is no replay and connection"

Step 6: Stop

CLIENT PROGRAM:

```
import java.io.*;
import java.net.*;
class pingdemo
{
        public static void  main(String args[])
       {
         BufferedReader in;
         try
         {
                 Runtime r=Runtime.getRuntime();
                 Process p=r.exec("Ping 127.0.0.1");

                 if(p==null)
                 System.out.println("could not connect");
                 in=new BufferedReader(new InputStreamReader(p.getInputStream()));
                 String line;

                 while((line=in.readLine())!=null)
                 {
                         if(line.startsWith("reply"))
                         System.out.println("this is reply");
                         else if(line.startsWith("request"))
                         System.out.println("there is no reply");
                         else if(line.startsWith("destinator"))
                         System.out.println("destinator host unreachabl");
                         else
                         System.out.println(line);
                 }
                 System.out.println(in.readLine());
                 in.close();
         } catch(IOException e)
         {System.out.println(e.toString());}
       }
}
```

OUTPUT:

```
MS-DOS Prompt
D:\java\bin>java pingdemo

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128
Reply from 127.0.0.1: bytes=32 time<10ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms
null
D:\java\bin>_
```

RESULT:

Thus the program for ping was implemented to check the connectivity of a host in a network.

EX. NO: 6

DATE: **IMPLEMENTATION OF ARP**

AIM:

To implement the concept of Address Resolution Protocol.

ALGORITHM:

Step 1: Start the process.

Step 2: Execute the server side program.

Step 3: Enter the MAC address of the machine.

Step 4: The IP address will be displayed.

Step 5: Execute the client side program.

Step 6: IP address will be received from server.

Step 7: Stop the process.

CLIENT SIDE:

```
import java.net.*;
import java.io.*;
public class arpclient
{  public static void main(String args[])throws IOException
 {        Socket s=new Socket("localhost",55);
          DataInputStream in=new DataInputStream(s.getInputStream());
           DataOutputStream out=new DataOutputStream(s.getOutputStream());
           String iparr[]={"10.0.1.45","172.16.5.21","172.16..5.22"};
          String macarr[]={"00-0c-6e-5c-3c-63","02-11-B6-F3-EF-21","03-12-B3-F3-EF-18"};
            String str=in.readLine();
          System.out.println("Ip Address received from server"+str);
           int flag=0;
           for(int i=0;i<5;i++)
          {
                    if(str.equals(iparr[i])==true)
                    {
                              flag=1;
                              String str1=macarr[i];
                              out.writeBytes(str1+"\n");
                              break;
                    }
```

```
        }
        if(flag==0)
        System.out.println("Given IPAddress is not in the network");
        s.close();
        }
}
```

SERVER SIDE:

```
import java.io.*;
import java.net.*;
class arpserver
{
 public static void main(String args[])throws IOException
 {
  ServerSocket ss=new ServerSocket(55);
  Socket s=ss.accept();
  DataInputStream in=new DataInputStream(s.getInputStream());
  DataOutputStream out=new DataOutputStream(s.getOutputStream());
  DataInputStream sysin=new DataInputStream(System.in);
  System.out.println("Enter an IP Address:");
  String str=sysin.readLine();
  out.writeBytes(str+"\n");
  System.out.println("The corresponding MAC address"+in.readLine());
  }
}
```

OUTPUT:

SERVER

E:\ss\exno7>javac arpserver.java

E:\ss\exno7>java arpserver

Enter an IP Address:

10.0.1.45

The corresponding MAC address is:00-0c-6e-5c-3c-63

CLIENT

E:\ss\exno7>javac arpclient.java

E:\ss\exno7>java arpclient

IP Address received from server

10.0.1.45

RESULT:

Thus the program for ARP has been executed successfully.

EX. NO: 7

DATE: **IMPLEMENATATION OF RARP**

AIM:

To implement the concept of R-Address Resolution Protocol.

ALGORITHM:

Step 1: Start the process.

Step 2: Execute the server side program.

Step 3: Enter the MAC address of the machine.

Step 4: The IP address will be displayed.

Step 5: Execute the client side program.

Step 6: MAC address will be received from server

Step 7: Stop the process

CLIENT

```
import java.net.*;
import java.io.*;
public class rarpc
{
 public static void main(String args[])throws IOException
 {
  Socket s=new Socket("localhost",55);
  DataInputStream in=new DataInputStream(s.getInputStream());
  DataOutputStream out=new DataOutputStream(s.getOutputStream());
  String iparr[]={"10.0.1.45","172.16.5.21","172.16..5.22"};
  String macarr[]={"00-0c-6e-5c-3c-63","02-11-B6-F3-EF-21","03-12-B3-F3-EF-18"};
  String str=in.readLine();
  System.out.println("Ip Address received from server"+str);
  int flag=0;
  for(int i=0;i<5;i++)
  {
   if(str.equals(macarr[i])==true)
   {
    flag=1;
    String str1=iparr[i];
    out.writeBytes(str1+"\n");
```

```
    break;
  }
}
if(flag==0)
System.out.println("Given IPAddress is not in the network");
s.close();
}
}
```

SERVER

```
import java.io.*;
import java.net.*;
class rarps
{
 public static void main(String args[])throws IOException
 {
  ServerSocket ss=new ServerSocket(55);
  Socket s=ss.accept();
  DataInputStream in=new DataInputStream(s.getInputStream());
  DataOutputStream out=new DataOutputStream(s.getOutputStream());
  DataInputStream sysin=new DataInputStream(System.in);
  System.out.println("Enter an mac Address:");
  String str=sysin.readLine();
  out.writeBytes(str+"\n");
  System.out.println("The corresponding MAC address"+in.readLine());
  }
```

}

OUTPUT:

SERVER

E:\ss\exno8>javac rarps.java

E:\ss\exno8>java rarps

Enter an MAC Address:

00-0c-6e-5c-3c-63

The corresponding IP address is:10.0.1.45

CLIENT

E:\ss\exno8>javac rarpc.java

E:\ss\exno8>java rarpc

MAC address received from server00-0c-6e-5c-3c-63

RESULT:

Thus the program for R-Address Resolution Protocol has been executed successfully.

EX. NO: 8

DATE: **IMPLEMENTATION OF FTP**

AIM:

To transfer the file from server to client using FTP protocol.

ALGORITHM:

Step 1: Start the process.

Step 2: create a file named output.

Step 3: Compile and run the server program.

Step 4: "File successfully sent" message will be displayed.

Step 5: Compile and run the client program.

Step 6: "File received successfully" message along with the copy of the file will be displayed.

Step 7: Stop the Process.

CLIENT PROGRAM:

```
import java.io.*;
import java.net.*;
public class Ftpclient
{
	public static void main(String a[])throws IOException
	{
		Socket s=new Socket(InetAddress.getLocalHost(),5555);
		DataInputStream s1=new DataInputStream(s.getInputStream());
		DataInputStream inp=new DataInputStream(System.in);
		DataOutputStream so=new DataOutputStream(s.getOutputStream());
		System.out.println("\n enter the filename(path)");
		String str=inp.readLine();
		so.writeBytes(str);
		so.writeBytes("\n");
		FileOutputStream fos=new FileOutputStream("output.txt");
		int str1;
		while((str1=s1.read())!=-1)
			fos.write((char)str1);
		System.out.println("\n file received successfully");
		s1.close();
		so.close();
```

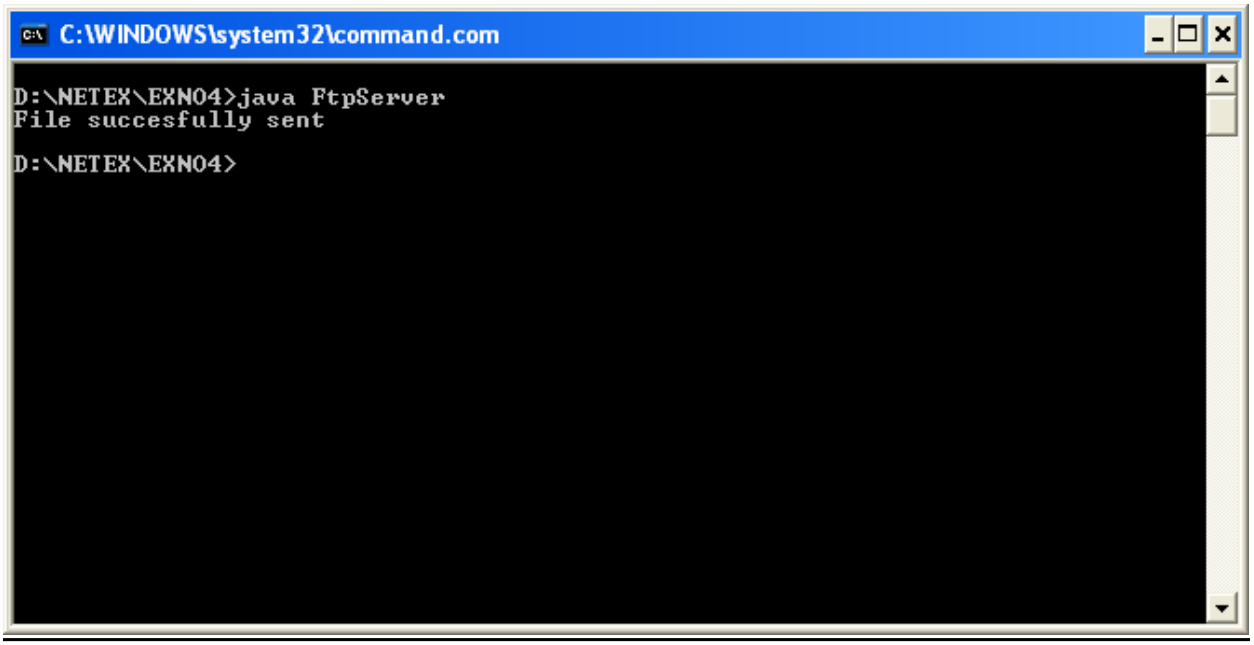
```
                inp.close();
                s.close();
        }
}
```

SERVER PROGRAM:

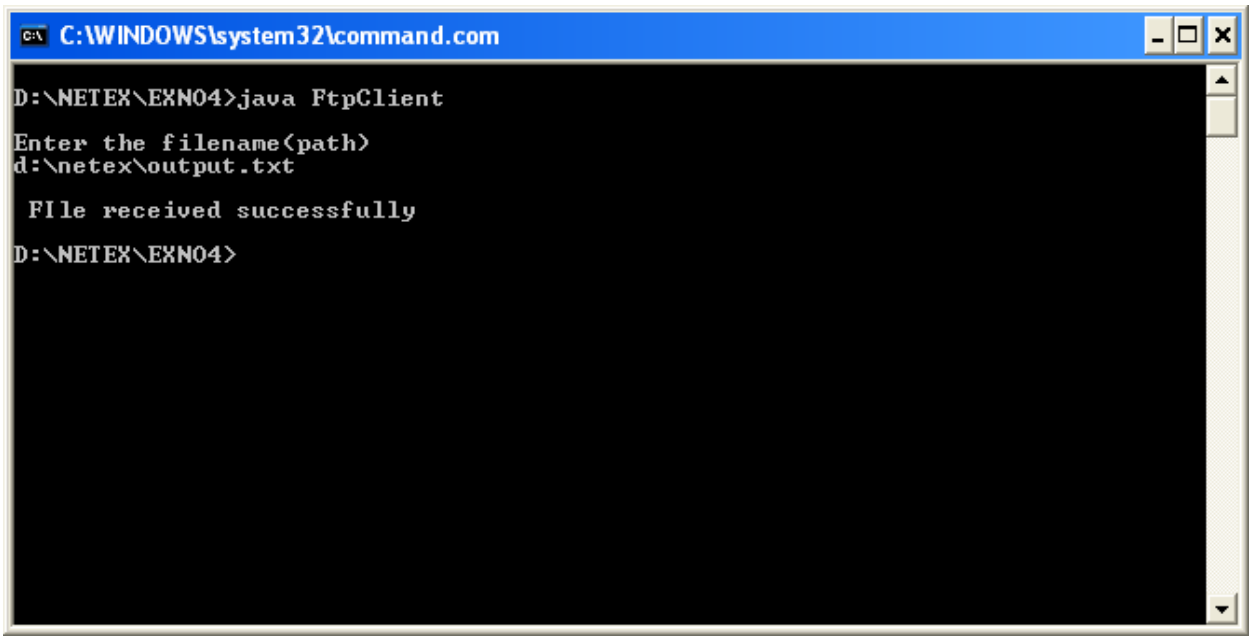
```
import java.io.*;
import java.net.*;
public class Ftpserver
{
        public static void main(String a[])throws IOException
        {
                ServerSocket ss=new ServerSocket(5555);

                Socket s=ss.accept();
                DataOutputStream dos=new DataOutputStream(s.getOutputStream());
                DataInputStream din=new DataInputStream(s.getInputStream());
                String s1;
                s1=din.readLine();
                FileInputStream fin=new FileInputStream(s1);
                int str1;
                while((str1=fin.read())!=-1)
                        dos.writeBytes(""+(char)str1);
                System.out.println("\n file successfully sent");
                dos.close();
                din.close();
                s.close();
```

```
        }
}
```

SERVER SIDE:



CLIENT SIDE:



RESULT:

Thus the file transfer through FTP has been executed successfully.

EX. NO: 9

DATE: IMPLEMENTATION OF REMOTE COMMAND EXECUTION

AIM:

To create a java program to implement the remote command execution.

ALGORITHM:

CLIENT SIDE

Step 1: Start the process.

Step 2: Connect the server host.

Step 3: Get the input

Step 4: Write the input to the output stream.

Step 5: Send the input to the server.

Step 6: Stop the process

SERVER SIDE

Step 1: Start the process.

Step 2: Connect to the client.

Step 3: Check whether the connection is accepted.

Step 4: Receive the input sent by the client.

Step 5: Print the data.

Step 6: Close the connection.

Step 7: Stop the process.

CLIENT SIDE

```
import java.net.*;
import java.io.*;
public class rmc
{       public static void main(String args[])
        {       try
                {
                        FileInputStream f=new FileInputStream("a.txt");
                         int si=f.available();
                        System.out.println("Size-----"+si);
                        System.out.println("contents in the file");
                         Socket s=new Socket("localhost",1500);
                        PrintStream p=new PrintStream(s.getOutputStream());
```

```
                    for(int i=0;i<si;i++)
                    {
                        char r=(char)f.read();
                        p.print(r);
                        System.out.print(r);
                    }
                    p.close();
                    s.close();
                }
                catch(Exception e)
                {       System.out.print(e);
                }
        }
}
```

SERVER SIDE:

```
import java.net.*;
import java.io.*;
public class rms
{       public static void main(String args[])
        {       int port=1500;
                ServerSocket server_socket;
                BufferedReader input;
                try{
                server_socket=new ServerSocket(port);
                System.out.println("Server waiting for client on
port"+server_socket.getLocalPort());
                while(true)
```

```
                {
                Socket socket=server_socket.accept();
                System.out.println("New connection
accepted"+socket.getInetAddress()+":"+socket.getPort());
                 input=new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                 while(true)
                {
                        String message=input.readLine();
                         Runtime r=Runtime.getRuntime();
                         Process p=r.exec(message);
                }
        }
}
        catch(Exception e)
        {       System.out.println(e);
        }
}
}
```

OUTPUT:

CLIENT

E:\ss\exno6>javac rmc.java

E:\ss\exno6>java rmc

NotePad

SERVER

E:\ss\exno6>javac rms.java

E:\ss\exno6>java rms

Notepad

RESULT:

Thus the program for remote command execution has been executed successfully.

EX. NO: 10

DATE **DISPLAY THE MESSAGE FROM CLIENT TO SERVER USING UDP**

AIM:

To write a program to transfer string between client and server using UDP.

ALGORITHM:

Step 1: Start the program

Step 2: Include the import java.net.*;

Import java.lang.*;Package

Step 3: Define a class for implementing main method.

Step 4: Port is the number that indicates what kind of protocol a server on the Internet is using.

Step 5: A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and noted.

Step 6: Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet.

Step 7: The string is transferred by client to server. If the server is ready it sends the sequence of the requested string to the client.

Step 8: Print out with the necessary details.

SERVER:

```
import java.io.*;
import java.net.* ;
import java.lang.*;
public class udpserver
{
public static int serverport=998;
public static int clientport=999;
public static int buffersize=1024;
public static byte buffer[]=new byte[buffersize];
public static DatagramSocket ds;
public static void serve() throws Exception
```

```
{       int pos=0;
        System.out.println(" I AM WAITING FOE YOUR INPUT");
        while(true)
        {       int c = System.in.read();
                switch(c)
                {
                case 1:
                System.out.println("server");
                return;
                case '\r':break;
                case '\n':ds.send(new
        DatagramPacket(buffer,pos,InetAddress.getLocalHost(),clientport));
                pos=0;
                break;
                default:
                buffer[pos++]=(byte)c;
                break;
                }
              }
        }
 public static void main(String arg[])throws Exception
{       ds=new DatagramSocket();
        serve();
}
}
```

CLIENT

```
import java.net.*;
class udpclient
{
public static int serverport=998;
public static int clientport=999;
public static int buffersize=1024;
public static byte buffer[]=new byte[buffersize];
public static DatagramSocket ds;
public static void serve() throws Exception
{
        while(true)
        {       DatagramPacket p = new DatagramPacket(buffer, buffer.length);
                ds.receive(p);
                System.out.println(new String(p.getData(),0,p.getLength()));
        }
}
 public static void main(String arg[])throws Exception
 {
        ds = new DatagramSocket(clientport);
        serve();
 }
}
```

OUTPUT:

```
D:\java\MEPRACTICE>java server
I AM WAITING FOR YOUR INPUT
Ibri college of technology

D:\java\MEPRACTICE>java client
Ibri college of technology
```