EX.NO: 06	APPLICATIONS USING TCP SOCKETS
DATE:	

1.FILE TRANSFER

Aim

To write a java program for file transfer using TCP Sockets.

Learning outcome:

After the completion of this experiment, student will be able to

☐ Develop a file transfer application by using TCP Socket

☐ Understand the reliability of TCP protocol.

☐ Understand how to transfer of computer files between a client and server on a computer network.

☐ Develop a client-server application by using TCP

Problem Statement:

To transfer computer files between a client and server on a computer network using TCP Sockets.

Concept:

Our aim is to send a file from the client machine to the server machine using TCP Communication.

System and Software tools required:

Package Required : Java Compiler
Operating System : UBUNTU
Minimum Requirement : Pentium III or Pentium IV with 2GB RAM 40 GB hard disk

Algorithm

Server

Step1: Import java packages and create class file server.
Step2: Create a new server socket and bind it to the port.

Step3: Accept the client connection
Step4: Get the file name and stored into the BufferedReader.
Step5: Create a new object class file and realine.
Step6: If file is exists then FileReader read the content until EOF is reached.
Step7: Stop the program.

Client

Step1: Import java packages and create class file server.
Step2: Create a new server socket and bind it to the port.
Step3: Now connection is established.
Step4: The object of a BufferReader class is used for storing data content which has been retrieved from socket object.
Step5: The content of file is displayed in the client window and the connection is closed.
Step6: Stop the program.

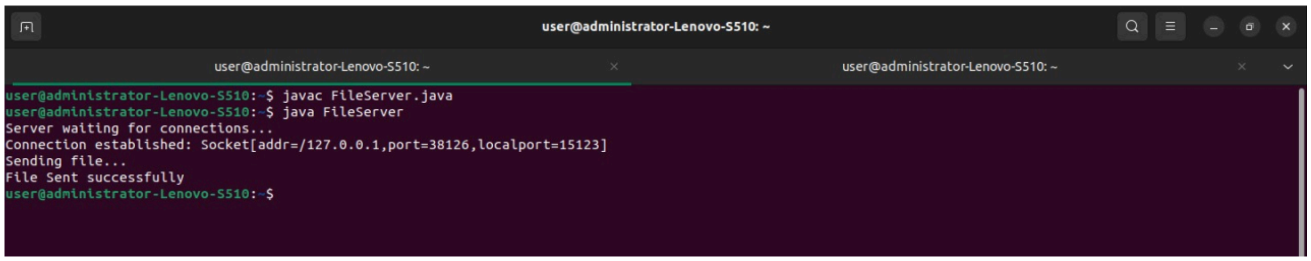
Execution of program:
Compiling the program: javac file name.java
Executing the program : java file name

SERVER

INPUT:

```
import java.io.*;

import java.net.ServerSocket;

import java.net.Socket;

class FileServer {

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

ServerSocket serverSocket = new ServerSocket(15123);

System.out.println("Server waiting for connections...");

Socket socket = serverSocket.accept();

System.out.println("Connection established: " + socket);

File transferFile = new File("lex.l");

byte[] bytearray = new byte[(int) transferFile.length()];
FileInputStream fin = new FileInputStream(transferFile);

BufferedInputStream bin = new BufferedInputStream(fin);

bin.read(bytearray, 0, bytearray.length);

OutputStream os = socket.getOutputStream();

System.out.println("Sending file...");

os.write(bytearray, 0, bytearray.length);

os.flush();

System.out.println("File Sent successfully");

socket.close();

serverSocket.close();

}}
```

OUTPUT:



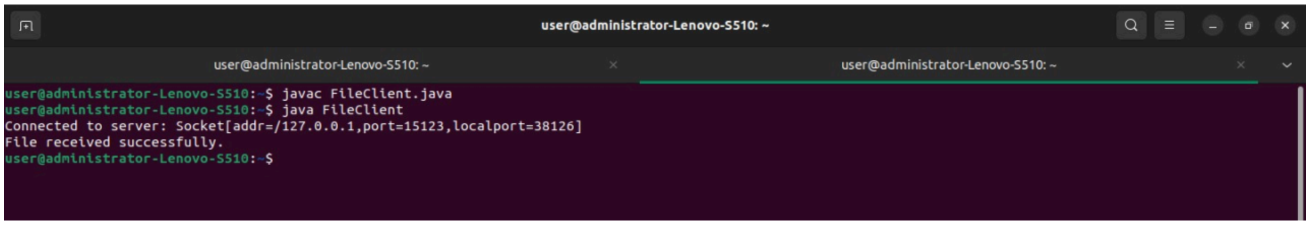
```
user@administrator-Lenovo-S510:~$ javac FileServer.java
user@administrator-Lenovo-S510:~$ java FileServer
Server waiting for connections...
Connection established: Socket[addr=/127.0.0.1,port=38126,localport=15123]
Sending file...
File Sent successfully
user@administrator-Lenovo-S510:~$
```

CLIENT

INPUT

```
import java.io.*;
import java.net.Socket;
public class FileClient {
    public static void main(String[] args) throws IOException {
        // Open connection to the server at port 15123 Socket
        socket = new Socket("127.0.0.1", 15123);
        System.out.println("Connected to server: " + socket);
        // File reception
        InputStream is = socket.getInputStream();
        // Enter the destination file name
        FileOutputStream fos = new FileOutputStream("lexa.l");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        // Buffer to read file chunks int
        bufferSize = 1024;
        byte[] bytearray = new byte[bufferSize];
        int bytesRead;
        // Read file data from the server
          while ((bytesRead = is.read(bytearray, 0, bufferSize)) != -1) {
            bos.write(bytearray, 0, bytesRead);
        }
        // Flush and close the output stream bos.flush();
        bos.close();
        System.out.println("File received successfully.");
        // Close resources
        socket.close();
      }}
```

OUTPUT:


user@administrator-Lenovo-S510: ~
user@administrator-Lenovo-S510: ~
user@administrator-Lenovo-S510: ~

```
user@administrator-Lenovo-S510:~$ javac FileClient.java
user@administrator-Lenovo-S510:~$ java FileClient
Connected to server: Socket[addr=/127.0.0.1,port=15123,localport=38126]
File received successfully.
user@administrator-Lenovo-S510:~$
```

2.REMOTE COMMAND EXECUTION

Objective:

To create a program for executing the remote command using TCP Socket.

Learing outcomes:

After the completion of this experiment, student will be able to

 Develop a program to execute remote the application by using TCP Socket.
Understand the reliability of TCP protocol.

- Understand how to execute remote commands in computer files between a client and server on a
computer network.

Problem Statement:

To create a program for executing a command in the system from a remote point.

Algorithm:

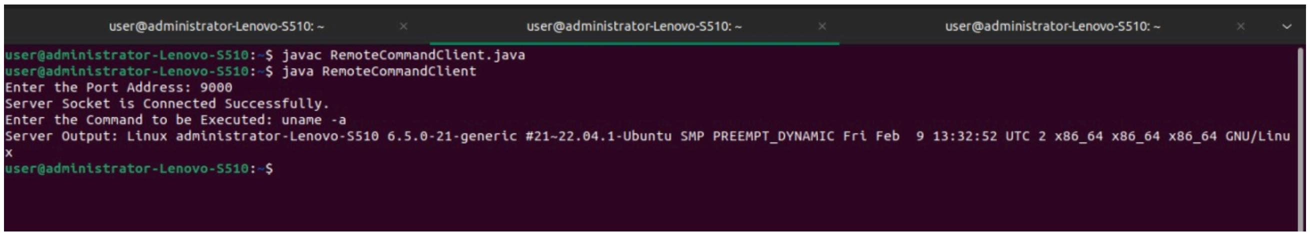
Step1: Start the program.
Step2: Create a server socket at the server side.
Step3: Create a socket at the client side and the connection is set to accept by the server socket using the
accept () method.
Step4: In the client side the remote command to be executed is given as input.
Step5: The command is obtained using the readLine () method of Buffer Reader.
Step6: Get the runtime object of the runtime class using getruntime ().
Step7: Execute the command using the exec () method of runtime.

2. SERVER

INPUT:

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket; import
java.util.Scanner;
class RemoteCommandServer {
  public static void main(String[] args) throws IOException { Scanner
      scanner = new Scanner(System.in); System.out.print("Enter the
      Port Address: ");
        int port = Integer.parseInt(scanner.nextLine());
            try (ServerSocket serverSocket = new ServerSocket(port)) {
                System.out.println("Server is Ready To Receive a Command.");
                System.out.println("Waiting...");
            // Wait and accept a connection
                try (Socket clientSocket = serverSocket.accept())
                    { if (clientSocket.isConnected()) {
                    System.out.println("Client Socket is Connected Successfully.");
                } InputStream in = clientSocket.getInputStream();
                OutputStream out = clientSocket.getOutputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(in)); PrintWriter
                writer = new PrintWriter(out, true);
                String cmd = reader.readLine();
                System.out.println("Received command: " + cmd);
                 ProcessBuilder processBuilder = new ProcessBuilder(cmd.split("\\s+"));
                Process process = processBuilder.start();
                // Read the command output and send it back to the client
                try (BufferedReader commandOutput = new BufferedReader(new
InputStreamReader(process.getInputStream()))) {
                  String outputLine;
                    while ((outputLine = commandOutput.readLine()) != null) {
                      writer.println(outputLine);
         }
        }
      }
   }
   }}
```

OUTPUT :



```
user@administrator-Lenovo-S510: ~        user@administrator-Lenovo-S510: ~        user@administrator-Lenovo-S510: ~
user@administrator-Lenovo-S510:~$ javac RemoteCommandClient.java
user@administrator-Lenovo-S510:~$ java RemoteCommandClient
Enter the Port Address: 9000
Server Socket is Connected Successfully.
Enter the Command to be Executed: uname -a
Server Output: Linux administrator-Lenovo-S510 6.5.0-21-generic #21~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Feb  9 13:32:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linu
x
user@administrator-Lenovo-S510:~$
```

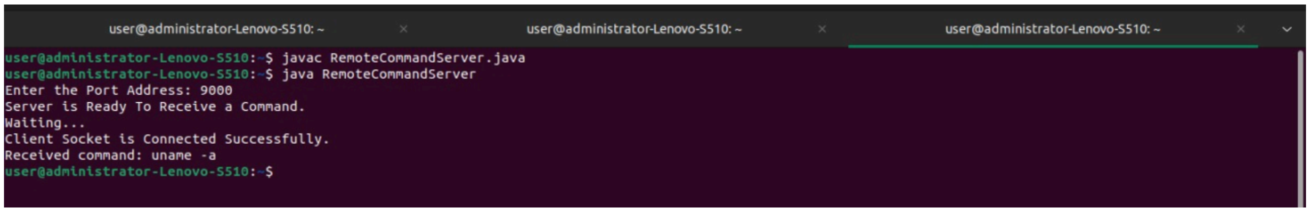
CLIENT

INPUT

```
import java.io.*;
import java.net.Socket; import
java.util.Scanner;
class RemoteCommandClient {
  public static void main(String[] args) throws IOException { Scanner
      scanner = new Scanner(System.in); System.out.print("Enter the
      Port Address: ");
```

```
        int port = Integer.parseInt(scanner.nextLine());
            try (Socket socket = new Socket("localhost", port))
                { if (socket.isConnected()) {
                System.out.println("Server Socket is Connected Successfully.");
            }
            InputStream in = socket.getInputStream();
              OutputStream out = socket.getOutputStream();
            BufferedReader userInputReader = new BufferedReader(new
 InputStreamReader(System.in));
            BufferedReader serverReader = new BufferedReader(new InputStreamReader(in));
            PrintWriter writer = new PrintWriter(out, true);
            System.out.print("Enter the Command to be Executed: "); String
            command = userInputReader.readLine();
            writer.println(command);
            // Read and print the command output received from the server String
            serverOutput;
                while ((serverOutput = serverReader.readLine()) != null) {
                    System.out.println("Server Output: " + serverOutput);
            }
  } }
 }
```

OUTPUT:


user@administrator-Lenovo-S510: ~
user@administrator-Lenovo-S510: ~
user@administrator-Lenovo-S510: ~

```
user@administrator-Lenovo-S510:~$ javac RemoteCommandServer.java
user@administrator-Lenovo-S510:~$ java RemoteCommandServer
Enter the Port Address: 9000
Server is Ready To Receive a Command.
Waiting...
Client Socket is Connected Successfully.
Received command: uname -a
user@administrator-Lenovo-S510:~$
```

3. CHAT

Objective:

To implement a CHAT application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages

.

Learning Outcomes:

After the completion of this experiment, student will be able to

- Develop a chat application by using TCP Socket
- Understand the reliability of TCP protocol.
- Understand how the reliable in order delivery is obtained using this protocol
- Develop a client-server application by using TCP

Problem Statement:

- A server program to establish the socket connection with the client for performing chat.
- A client program which on establishing a connection with the server for performing chat.

Concept:

1. It uses TCP socket communication .We have a server as well as a client.
2. Both can be run in the same machine or different machines. If both are running in the machine, the address to be given at the client side is local host address.
3. If both are running in different machines, then in the client side we need to specify the ip address of machine in which server application is running.

Algorithm:

Server

Step1: Start the program and create server and client sockets.
Step2: Use input streams to get the message from user.
Step3: Use output streams to send message to the client.
Step4: Wait for client to display this message and write a new one to be displayed by the server.
Step5: Display message given at client using input streams read from socket.
Step6: Stop the program.

Client

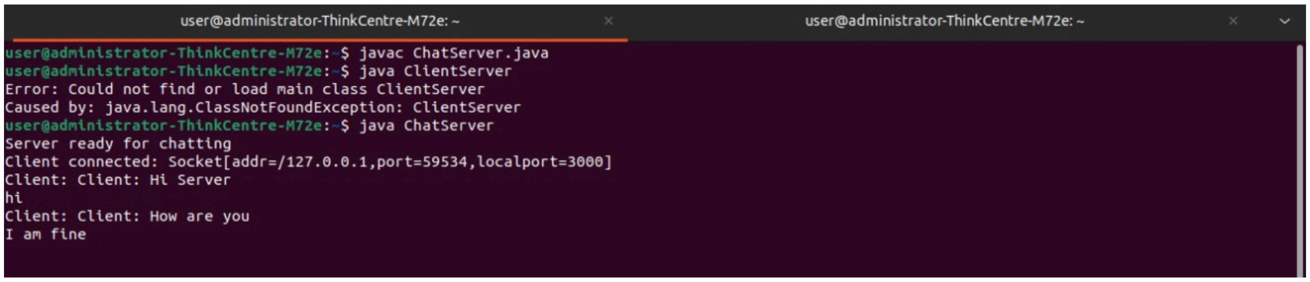
Step1: Start the program and create a client socket that connects to the required host and port.
Step2: Use input streams read message given by server and print it.
Step3: Use input streams; get the message from user to be given to the server.
Step4: Use output streams to write message to the server.
Step5: Stop the program.

<u>SERVER</u>

INPUT:

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
class ChatServer {
 public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(3000);
    System.out.println("Server ready for chatting");
     Socket clientSocket = serverSocket.accept();
     System.out.println("Client connected: " + clientSocket);
     // Reading from keyboard
     BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
     // Sending to client
     OutputStream ostream = clientSocket.getOutputStream();
     PrintWriter pwrite = new PrintWriter(ostream, true);
     // Receiving from client
     InputStream istream = clientSocket.getInputStream();
     BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream)); String
     receiveMessage, sendMessage;
     while (true) {
        // Receive message from the client
           if ((receiveMessage = receiveRead.readLine()) != null)
              { System.out.println("Client: " + receiveMessage);
        }
        // Read message from the server's console
        sendMessage = keyRead.readLine();
        // Send the message to the client
        pwrite.println("Server: " + sendMessage);
        pwrite.flush();
     } }}
```

OUTPUT:


user@administrator-ThinkCentre-M72e: ~
user@administrator-ThinkCentre-M72e: ~

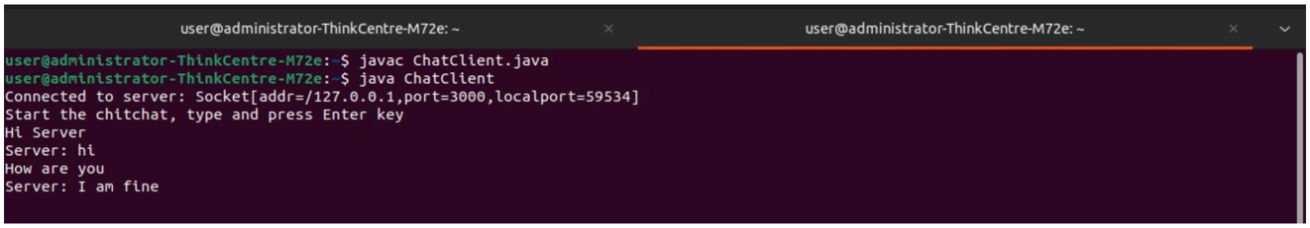
```
user@administrator-ThinkCentre-M72e:~$ javac ChatServer.java
user@administrator-ThinkCentre-M72e:~$ java ClientServer
Error: Could not find or load main class ClientServer
Caused by: java.lang.ClassNotFoundException: ClientServer
user@administrator-ThinkCentre-M72e:~$ java ChatServer
Server ready for chatting
Client connected: Socket[addr=/127.0.0.1,port=59534,localport=3000]
Client: Client: Hi Server
hi
Client: Client: How are you
I am fine
```

<u>CLIENT</u>

INPUT:

```
import java.io.*;
import java.net.Socket;
class ChatClient {
  public static void main(String[] args) throws IOException { Socket
      socket = new Socket("127.0.0.1", 3000);
      System.out.println("Connected to server: " + socket);
        // Reading from keyboard
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // Sending to server
        OutputStream ostream = socket.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        // Receiving from server
        InputStream istream = socket.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream)); String
        receiveMessage, sendMessage;
        System.out.println("Start the chitchat, type and press Enter key");
        while (true) {
            // Read message from the client's console
            sendMessage = keyRead.readLine();
            // Send the message to the server
            pwrite.println("Client: " + sendMessage);
            pwrite.flush();
            // Receive message from the server
                if ((receiveMessage = receiveRead.readLine()) != null) {
                    System.out.println(receiveMessage);
            } } }}
```

OUTPUT:



```
user@administrator-ThinkCentre-M72e:~$ javac ChatClient.java
user@administrator-ThinkCentre-M72e:~$ java ChatClient
Connected to server: Socket[addr=/127.0.0.1,port=3000,localport=59534]
Start the chitchat, type and press Enter key
Hi Server
Server: hi
How are you
Server: I am fine
```

4. CONCURRENT SERVER

Objective:

To write a program for concurrent communication of client to the server using TCP Socket.

Learning Outcomes:

After the completion of this experiment, student will be able to

- ☐ Understand the reliability of TCP protocol.
- ☐ Understand how a a server can handle multiple clients at the same time in parallel
- ☐ Develop a client-server application by using TCP

Problem Statement:

To have a reliable concurrent communication between client and the server.

Concept:

Concurrent Server: The server can be iterative, i.e. it iterates through each client and serves one request at a time. Alternatively, a server can handle multiple clients at the same time in parallel, and this type of a server is called a concurrent server.

Algorithm:

Server

Step 1: Create a socket and bind to the address. Leave socket unconnected.
Step 2 : Leave socket in passive mode, making it ready for use by a server.
Step 3: Repeatedly call accept to receive the next request from a client to handle the response with the through socket.

Client

Step 1: Begin with a connection passed from the server (i.e., a socket for the connection).
Step 2: Use input streams; get the message from user to be given to the server.
Step 3: Use input streams read message given by server and print it.
Step 4: Use output streams to write message to the server.
Step 5: Close the connection and exit, i.e., slave terminates after handling all requests from one client.

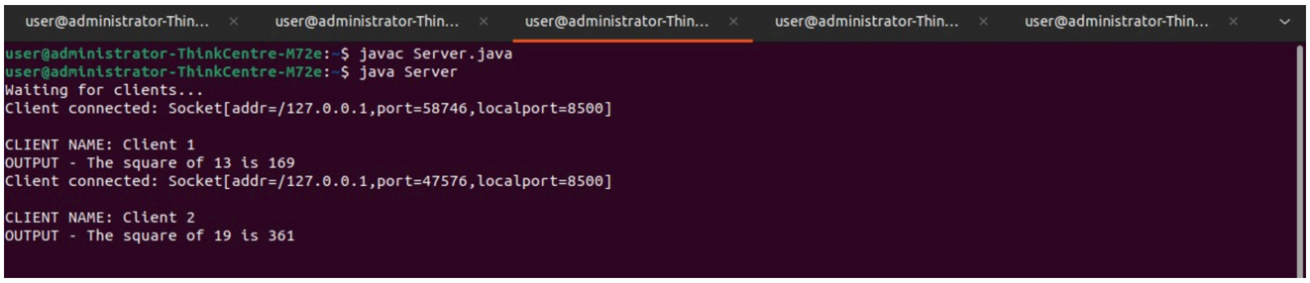
<u>SERVER</u>

INPUT:

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class Server {
  public static void main(String[] args) throws IOException {
     ServerSocket ss = new ServerSocket(8500);
     System.out.println("Waiting for clients...");

       // Using a thread pool to handle multiple clients concurrently ExecutorService
       executorService = Executors.newFixedThreadPool(5);

       while (true) {
           Socket clientSocket = ss.accept();
           System.out.println("Client connected: " + clientSocket);

           // Create a new thread for each client
           executorService.execute(new ClientHandler(clientSocket));
       }
   }
}

   class ClientHandler implements Runnable {
       private final Socket clientSocket;

  public ClientHandler(Socket clientSocket) {
     this.clientSocket = clientSocket;
   }

   @Override public
   void run() {
       try {
           BufferedReader br = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

```
            String clientName = br.readLine();
            System.out.println("\nCLIENT NAME: " + clientName);

            int num = Integer.parseInt(br.readLine());

            int square = num * num;

            PrintWriter pw = new PrintWriter(clientSocket.getOutputStream(), true);
            pw.println(square);

            System.out.println("OUTPUT - The square of " + num + " is " + square);

            // Close the client socket clientSocket.close();
            } catch (IOException e)
              {
              e.printStackTrace();
          }
      }
  }
```

OUTPUT:


user@administrator-Thin...
user@administrator-Thin...
user@administrator-Thin...
user@administrator-Thin...
user@administrator-Thin...

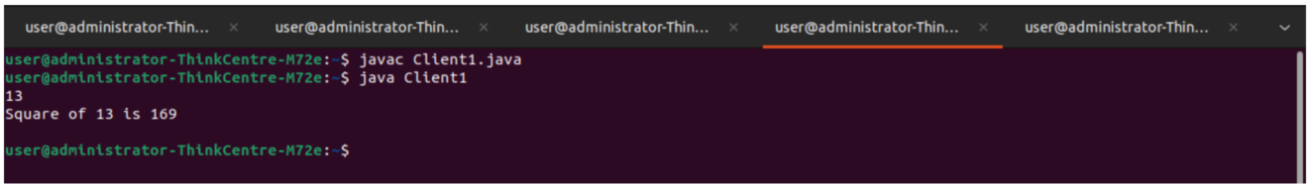
```
user@administrator-ThinkCentre-M72e:~$ javac Server.java
user@administrator-ThinkCentre-M72e:~$ java Server
Waiting for clients...
Client connected: Socket[addr=/127.0.0.1,port=58746,localport=8500]

CLIENT NAME: Client 1
OUTPUT - The square of 13 is 169
Client connected: Socket[addr=/127.0.0.1,port=47576,localport=8500]

CLIENT NAME: Client 2
OUTPUT - The square of 19 is 361
```

<u>CLIENT 1</u>

INPUT:

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

public class Client1 {
  public static void main(String[] args) throws IOException { Socket
     socket = new Socket("localhost", 8500);

      BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); int
      num = Integer.parseInt(br.readLine());

      PrintWriter pw = new PrintWriter(socket.getOutputStream(), true);
      pw.println("Client 1");
      pw.println(num);

      BufferedReader br1 = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      int square = Integer.parseInt(br1.readLine());
      System.out.println("Square of " + num + " is " + square + "\n");

      // Close the socket
      socket.close();
   }
}
```

OUTPUT:



```
user@administrator-ThinkCentre-M72e:~$ javac Client1.java
user@administrator-ThinkCentre-M72e:~$ java Client1
13
Square of 13 is 169

user@administrator-ThinkCentre-M72e:~$
```

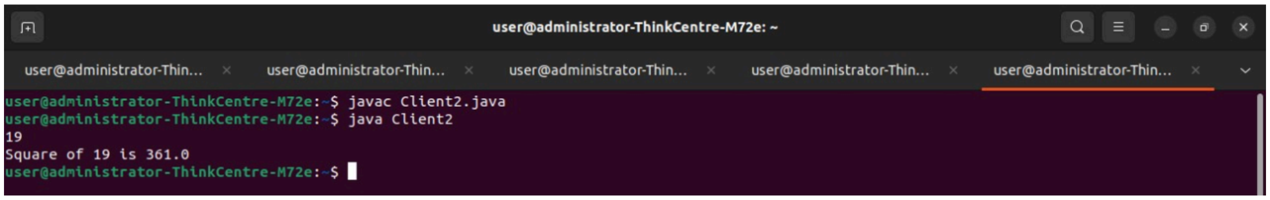
<u>CLIENT 2</u>

INPUT:

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

public class Client2 {
  public static void main(String[] args) throws IOException { Socket
     s = new Socket("localhost", 8500);
```

```
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); int

        num = Integer.parseInt(br.readLine());

        PrintWriter pw = new PrintWriter(s.getOutputStream(), true);
        pw.println("Client 2");
        pw.println(num);

        BufferedReader br1 = new BufferedReader(new InputStreamReader(s.getInputStream()));
        double squareRoot = Double.parseDouble(br1.readLine());
        System.out.println("Square of " + num + " is " + squareRoot);

        s.close();
    }
}
```

OUTPUT:



```
user@administrator-ThinkCentre-M72e:~$ javac Client2.java
user@administrator-ThinkCentre-M72e:~$ java Client2
19
Square of 19 is 361.0
user@administrator-ThinkCentre-M72e:~$
```