Name: Sushant Bagul

Roll No : 221070011

Batch-A

Experiment No. 4

Aim : Study of Socket Programming and Client–Server model.

Theory :

What is Socket ?

A socket serves as an endpoint for communication between two nodes or machines within a network, uniquely identified by an IP address and a port number. Client-side programming encompasses all the programming activities performed on the client side of this communication link. In client-server communication, the fundamental process begins with the client waiting for the server to start. Once the server is active, the client initiates a request through the connection established between their respective sockets. Subsequently, the client awaits a response from the server.

Establish a Socket Connection

For communication between two machines, sockets must be present at both endpoints. The client needs to know the IP address of the server machine to establish a connection. A socket can be created with a single line of code:

Socket clientSocket = new Socket("127.0.0.1", 4000);

Here, clientSocket represents the socket at the client side, 4000 is the TCP port number, and 127.0.0.1 is the IP address of the server machine.

Communication

Once sockets are successfully created, communication between them is facilitated using data streams. In networking, there are two types of data streams: the Input Stream and the Output Stream. The server's input stream is connected to the client's output stream, while the client's input stream is connected to the server's output stream.

Closing the Connection

Finally, once the purpose is fulfilled the programmer needs to close the connection established earlier

Server Programming

Server runs on a machine that is present over a network and bound to a particular port number.

Establish a Socket Connection

Using the Socket class in java, one can create a socket with a single line of code:

ServerSocket serverSocket = new ServerSocket(4000);

Where the argument 4000 is our port number and the server waits for a connection request from the client side at the server socket. Once a connection request is received then the below line of code gets executed.

Socket clientSocket = serverSocket.accept();

This means that if the protocol is maintained and followed properly, then the request would be accepted by the server.

Close the Connection

Finally, once the purpose is fulfilled the client sends a request to the server to terminate the connection between the client socket and server socket.

Code & Output :

Program 1 :

Client.java

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

public class Client {
   // Initialize socket and input/output streams
   private Socket socket = null;
   private DataInputStream input = null;
   private DataOutputStream out = null;

   // Constructor to specify import java.io.*;
import java.net.*;

class MyServer1 {
   public static void main(String[] args) throws Exception {
       // Create a server socket listening on port 3333
       ServerSocket ss = new ServerSocket(3333);
       System.out.println("Server is listening on port 3333");

       // Accept client connections
       Socket s = ss.accept();
       System.out.println("Client connected");

       // Create input and output streams
       DataInputStream din = new DataInputStream(s.getInputStream());
       DataOutputStream dout = new DataOutputStream(s.getOutputStream());
       BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

       String str = "", str2 = "";

       // Keep communicating until "BYE" is received
       while (!str.equals("BYE")) {
           str = din.readUTF();
           System.out.println("Client says: " + str);
```

```
            str2 = br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }

        // Close resources
        din.close();
        dout.close();
        s.close();
        ss.close();
    }
}
IP address and port
    public Client(String address, int port) {
        // Establish a connection
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // Takes input from terminal
            input = new DataInputStream(System.in);

            // Sends output to the socket
            out = new DataOutputStream(socket.getOutputStream());
        } catch (UnknownHostException u) {
            System.out.println("Unknown host: " + u.getMessage());
            return;
        } catch (IOException i) {
            System.out.println("IO error: " + i.getMessage());
            return;
        }

        // String to read message from input
        String line = "";

        // Keep reading until "BYE" is input
        while (!line.equals("BYE")) {
            try {
                line = input.readLine();
                out.writeUTF(line);
```

```
            } catch (IOException i) {
                System.out.println("IO error: " + i.getMessage());
            }
        }

        // Close the connection
        try {
            input.close();
            out.close();
            socket.close();
        } catch (IOException i) {
            System.out.println("IO error: " + i.getMessage());
        }
    }

    public static void main(String[] args) {
        new Client("127.0.0.1", 5000);
    }
}
```

Server.java

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

public class Server1 {
    // Initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;

    // Constructor with port
    public Server1(int port) {
        // Starts server and waits for a connection
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");
```

```
            socket = server.accept();
            System.out.println("Client accepted");

            // Takes input from the client socket
            in = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));

            String line = "";

            // Reads message from client until "BYE" is sent
            while (!line.equals("BYE")) {
                try {
                    line = in.readUTF();
                    System.out.println(line);
                } catch (IOException i) {
                    System.out.println(i);
                }
            }
            System.out.println("Closing connection");

            // Close connection
            socket.close();
            in.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }

    public static void main(String[] args) {
        new Server1(5000);
    }
}
```

```
^Csysadmin@sysadmin:~/Documents/CN$ java client.java
client.java:42: warning: [deprecation] readLine() in DataInputStream has been deprecated
line = input.readLine();
           ^
1 warning
Connected
Hello from client
```

```
^Csysadmin@sysadmin:~/Documents/CN$ java Server1.java
Server started
Waiting for a client ...
Client accepted
Hello from client
hello from server
```

Program 2:

Client2.java

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

class MyClient1 {
   public static void main(String[] args) throws Exception {
       // Create a socket connection to the server
       Socket s = new Socket("localhost", 3333);

       // Create input and output streams
       DataInputStream din = new DataInputStream(s.getInputStream());
       DataOutputStream dout = new DataOutputStream(s.getOutputStream());
       BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

       String str = "", str2 = "";

       // Keep communicating until "BYE" is sent
       while (!str.equals("BYE")) {
           str = br.readLine();
           dout.writeUTF(str);
           dout.flush();
           str2 = din.readUTF();
           System.out.println("Server says: " + str2);
```

```
        }

        // Close resources
        dout.close();
        s.close();
    }
```

Server2.java

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

class MyServer1 {
    public static void main(String[] args) throws Exception {
        // Create a server socket listening on port 3333
        ServerSocket ss = new ServerSocket(3333);
        System.out.println("Server is listening on port 3333");

        // Accept client connections
        Socket s = ss.accept();
        System.out.println("Client connected");

        // Create input and output streams
        DataInputStream din = new DataInputStream(s.getInputStream());
        DataOutputStream dout = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

        String str = "", str2 = "";

        // Keep communicating until "BYE" is received
        while (!str.equals("BYE")) {
            str = din.readUTF();
            System.out.println("Client says: " + str);
            str2 = br.readLine();
            dout.writeUTF(str2);
            dout.flush();
        }
```

```
        // Close resources
        din.close();
        dout.close();
        s.close();
        ss.close();
    }
}
```

```
sysadmin@sysadmin:~/Documents/CN$ java MyClient1.java
Hello from client
Server says: hello from server
```

```
sysadmin@sysadmin:~/Documents/CN$ java MyServer1.java
Client says: Hello from client
hello from server
```

Conclusion :

In this experiment, we understood the need for sockets and how to implement them in a Java program. We also implemented two programs using sockets that help establish a connection between a client and a server.