

Name: Aditya Rajesh Sawant

Subject: CN

Roll No: 46

Division: TE4

Batch: D

Experiment: 8

Aim: Java socket programming using TCP or UDP

Theory:

- Java Socket programming is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming. The client in socket programming must know two information:

a. IP Address of Server, and

b. Port number. Here, we are going to make one-way client and server communication. In this

application, the client sends a message to the server, the server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate between client and server. Through this class, we can read and write messages. The ServerSocket class is

used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of the client, it returns the instance of Socket at server-side. #Socket class A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

#ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients. Creating Server:

To create the server application, we need to create an instance of the ServerSocket

class. Here, we are using 8888 port number for the communication between the client and server. You may also choose any other port number. The accept() method

waits for the client. If a client connects with the given port number, it returns an instance of Socket. ServerSocket ss=new ServerSocket(8888);

Socket s=ss.accept();//establishes connection and waits for the client

Creating Client:

To create the client application, we need to create an instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on the same system. Socket s=new Socket("localhost",8888);

Server Side:

```
package sakec;
```

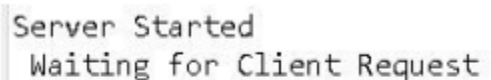
```
import java.io.*;
```

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

```
public class ServerSoc {
```

```
public static void main (String args[])throws Exception
{
    System.out.println("Server Started");
    ServerSocket ss = new ServerSocket (8888);
    System.out.println(" Waiting for Client Request ");
    Socket s = ss.accept();
    System.out.println(" Connected Client ");
    BufferedReader br = new BufferedReader(new
    InputStreamReader(s.getInputStream()));
    String str = br.readLine();
    System.out.println("Client Data : " + str);
}
}
```

OUTPUT:

A screenshot of a terminal window showing the output of the server program. The text "Server Started" is on the first line, and "Waiting for Client Request" is on the second line. The text is displayed in a monospaced font with a light gray background.

```
Server Started
Waiting for Client Request
```

CLIENT Side:

```
package sakec;

import java.util.*;
```

```
import java.net.*;
import java.io.*;
public class ClientSoc {
    public static void main( String args[] ) throws Exception
    {
        Scanner input = new Scanner(System.in);
        String ip = "localhost";
        int port = 8888;
        Socket client_soc = new Socket(ip, port) ;
        String msg = " I am client\n";
        OutputStreamWriter os = new
        OutputStreamWriter(client_soc.getOutputStream()) ;
        PrintWriter out = new PrintWriter(os) ;
        os.write(msg) ;
        os.flush() ;
    }
}
```

OUTPUT:

```
Server Started
Waiting for Client Request
Connected Client
Client Data : I am client
|
```