Assignment No:4
Name:Prathmesh Pattewar
Prn:12420193

Title: Client server programs using TCP Berkeley socket.

Problem Statement: Write the client server programs using TCP Berkeley socket primitives for wired /wireless network for following a. to say Hello to Each other b. File transfer.

Course Objective: To learn transport layer and application layer protocols used in the Internet.

Course Outcome: Develop Client-Servers by the means of correct standards, protocols and technologies.

Tools Required: Eclipse, Java.

Theory:

1. **Introduction**

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.
The java.net package provides support for the two common network protocols:

- **TCP:** TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP:** UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

2. **Berkeley Socket Primitives (TCP/IP Socket Programming):**

The Berkeley Sockets API was developed at the University of California, Berkeley, and forms the foundation for network communication in many operating systems including UNIX, Linux, and Windows.

Socket Primitives for TCP (Connection-Oriented):

Primitive	**Description**
socket()	Creates a new socket descriptor (endpoint for communication).
bind()	Binds the socket to a local IP address and port.
listen()	Marks the socket as a passive socket to accept incoming connection requests.
accept()	Accepts a connection request from a client and returns a new socket descriptor for communication.
connect()	Used by the client to initiate a connection with the server.

Primitive	Description
send() / write()	Sends data over the connection.
recv() / read()	Receives data from the connection.
close()	Closes the socket and terminates the connection.

3. **Socket Programming Flow (TCP):**

Server Side:

- **Create socket** using socket()
- **Bind** the socket to an IP and port using bind()
- **Listen** for connections using listen()
- **Accept** incoming client connections using accept()
- **Communicate** using read() and write()
- **Close** the connection with close()

lient Side:

- **Create socket** using socket()
- **Connect** to server using connect()
- **Communicate** using read() and write()
- **Close** the connection with close()

4. **Socket Programming Using Java:**

Sockets provide the communication mechanism between two computers using TCP. 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 server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.

- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.
TCP is a two way communication protocol, so data can be sent across both streams at the same time. There are following use full classes providing complete set of methods to implement sockets.

5. ServerSocket Class Methods:

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.
public ServerSocket(int port, int backlog) throws IOException Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
public ServerSocket(int port, int backlog, InetAddress address) throws IOException Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on
public ServerSocket() throws IOException Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket
public Socket accept() throws IOException Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely

6. Socket Class Methods:

The **java.net.Socket** class represents the socket that both the client and 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.

public Socket(String host, int port) throws UnknownHostException, IOException.

This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
public Socket(InetAddress host, int port) throws IOException This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
public void connect(SocketAddress host, int timeout) throws IOException **This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.**
public InetAddress getInetAddress() This method returns the address of the other computer that this socket is connected to.
public int getPort() Returns the port the socket is bound to on the remote machine.
public void close() throws IOException Closes the socket, which makes this Socket object no longer capable of connecting again to any server

Program:

//*************************Client.Java************************************

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

public class Client {
    public static void main(String[] args) {
        String host = "192.168.185.75";
        int port = 5000;

        try (
            Socket socket = new Socket(host, port);
            DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());
            DataInputStream dis = new
DataInputStream(socket.getInputStream());
            Scanner scanner = new Scanner(System.in);
        ) {
            System.out.println("Connected to server.");
            System.out.println("Send (1) Message or (2)
File?");
```

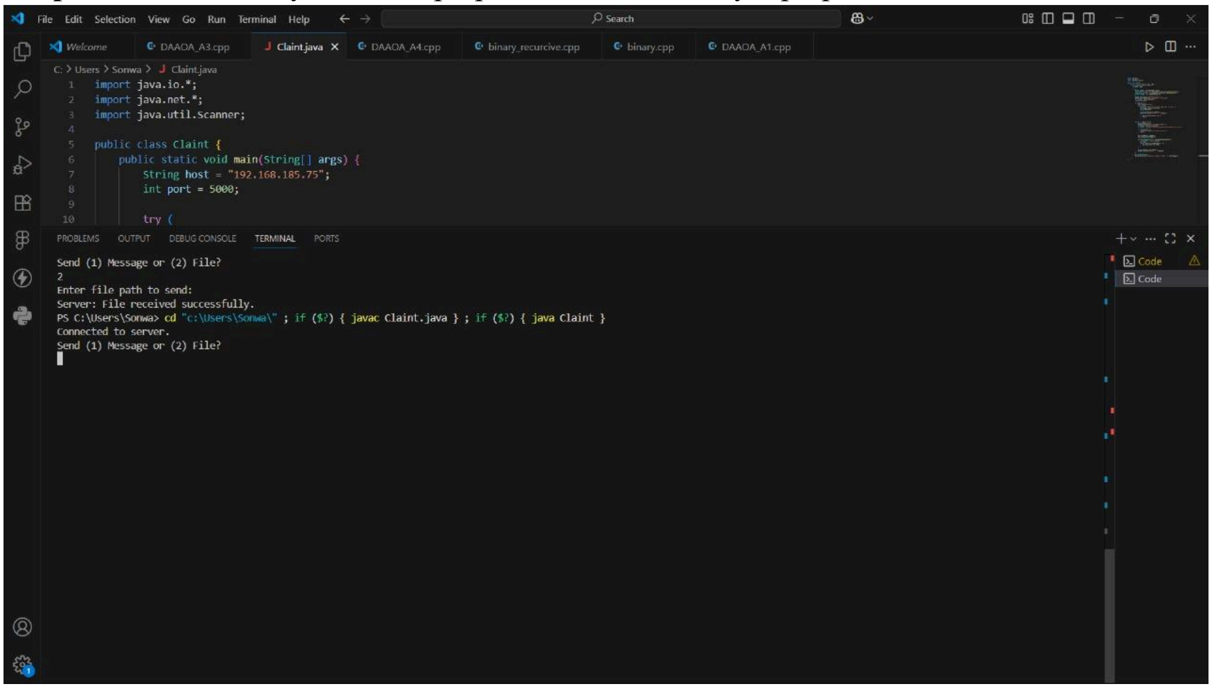
```
            int choice = scanner.nextInt();
            scanner.nextLine();

            if (choice == 1) {
                dos.writeUTF("message");
                while (true) {
                    System.out.print("Enter message (type 'bye' to exit): ");
                    String msg = scanner.nextLine();
                    dos.writeUTF(msg);

                    String response = dis.readUTF();
                    System.out.println("Server: " + response);

                    if (msg.equalsIgnoreCase("bye")) {
                        break;
                    }
                }

            } else if (choice == 2) {
                dos.writeUTF("file");
                System.out.print("Enter file path to send: ");
                String path = scanner.nextLine();
                File file = new File(path);

                if (!file.exists()) {
                    System.out.println("File does not exist.");
                    return;
                }

                dos.writeUTF(file.getName());
                dos.writeLong(file.length());

                try (FileInputStream fis = new FileInputStream(file)) {
                    byte[] buffer = new byte[4096];
                    int read;
                    while ((read = fis.read(buffer)) > 0) {
                        dos.write(buffer, 0, read);
                    }
                }

                String response = dis.readUTF();
                System.out.println("Server: " + response);
            }

        } catch (IOException e) {
            System.out.println("Connection failed or closed: "
+ e.getMessage());
        }
    }
}
```

//*************************Server.Java*********************************

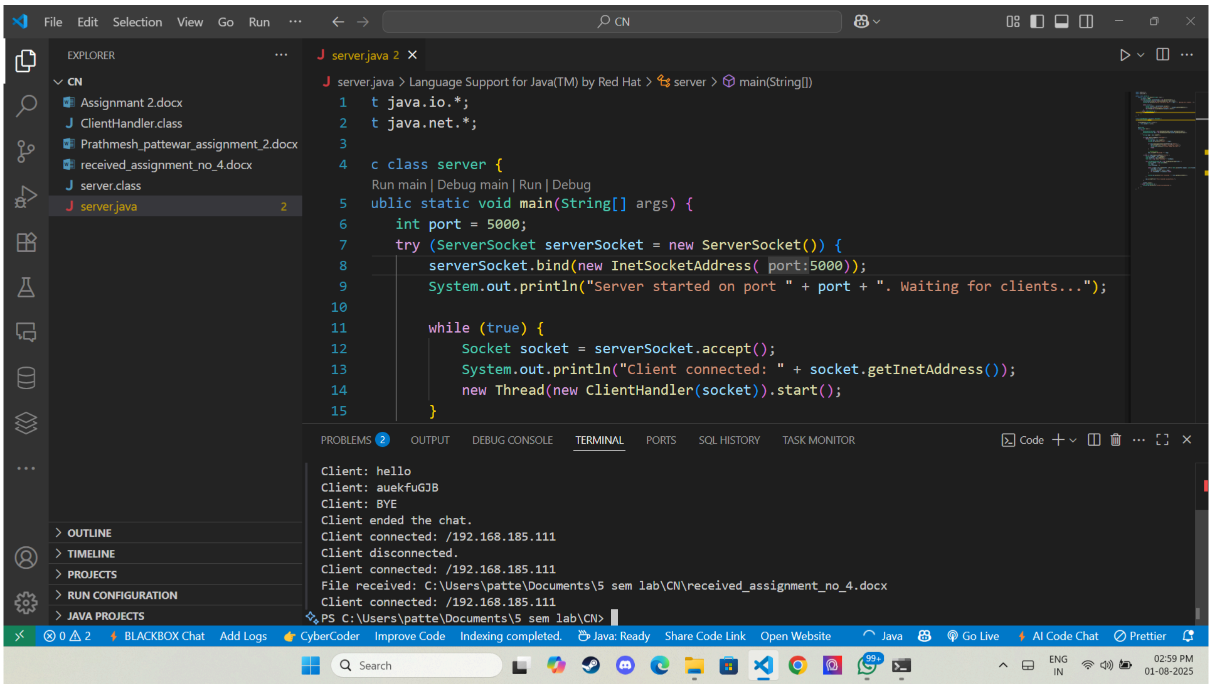
```
import java.io.*;
import java.net.*;

public class server {
    public static void main(String[] args) {
        int port = 5000;
        try (ServerSocket serverSocket = new ServerSocket()) {
            serverSocket.bind(new InetSocketAddress( 5000));
            System.out.println("Server started on port " + port
+ ". Waiting for clients...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected: " +
socket.getInetAddress());
                new Thread(new ClientHandler(socket)).start();
            }
```

```java
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler implements Runnable {
    private Socket socket;

    ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try (
            DataInputStream dis = new
DataInputStream(socket.getInputStream());
            DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());
        ) {
            String type = dis.readUTF();

            if (type.equals("message")) {
                while (true) {
                    String msg = dis.readUTF();
                    System.out.println("Client: " + msg);

                    if (msg.equalsIgnoreCase("bye")) {
                        dos.writeUTF("Goodbye! Connection
closed.");
                        System.out.println("Client ended the
chat.");
                        break;
                    }

                    // Respond back
                    dos.writeUTF("Received: " + msg);
                }
            } else if (type.equals("file")) {
                String fileName = dis.readUTF();
                long fileSize = dis.readLong();
                File file = new File("received_" + fileName);

                try (FileOutputStream fos = new
FileOutputStream(file)) {
                    byte[] buffer = new byte[4096];
                    int read;
                    long totalRead = 0;

                    while ((read = dis.read(buffer, 0,
Math.min(buffer.length, (int)(fileSize - totalRead)))) > 0) {
                        totalRead += read;
                        fos.write(buffer, 0, read);
                        if (totalRead >= fileSize) break;
                    }

                    System.out.println("File received: " +
file.getAbsolutePath());
                }

                dos.writeUTF("File received successfully.");
            }

            socket.close();
        } catch (IOException e) {
            System.out.println("Client disconnected.");
        }
    }
}
```

Output: Client is on my friends laptop and server is on my laptop.


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

public class Claint {
    public static void main(String[] args) {
        String host = "192.168.185.75";
        int port = 5000;

        try (
```

```
Send (1) Message or (2) File?
2
Enter file path to send:
Server: File received successfully.
PS C:\Users\Sonwa> cd "c:\Users\Sonwa\" ; if ($?) { javac Claint.java } ; if ($?) { java Claint }
Connected to server.
Send (1) Message or (2) File?
```


```java
t java.io.*;
t java.net.*;

c class server {
ublic static void main(String[] args) {
    int port = 5000;
    try (ServerSocket serverSocket = new ServerSocket()) {
        serverSocket.bind(new InetSocketAddress( port:5000));
        System.out.println("Server started on port " + port + ". Waiting for clients...");

        while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("Client connected: " + socket.getInetAddress());
            new Thread(new ClientHandler(socket)).start();
        }
```

```
Client: hello
Client: auekfuGJB
Client: BYE
Client ended the chat.
Client connected: /192.168.185.111
Client disconnected.
Client connected: /192.168.185.111
File received: C:\Users\patte\Documents\5 sem lab\CN\received_assignment_no_4.docx
Client connected: /192.168.185.111
PS C:\Users\patte\Documents\5 sem lab\CN>
```

Conclusion: The client-server programs using TCP Berkeley socket primitives demonstrate the fundamental principles of reliable communication over wired or wireless networks. By utilizing TCP sockets, these programs ensure a connection-oriented, reliable, and ordered data transfer between client and server.