Experiment: 6

To implement chat server using socket programming (using UDP & TCP)

Aim: Write a program to implement Chat server using socket programming

Introduction:

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:

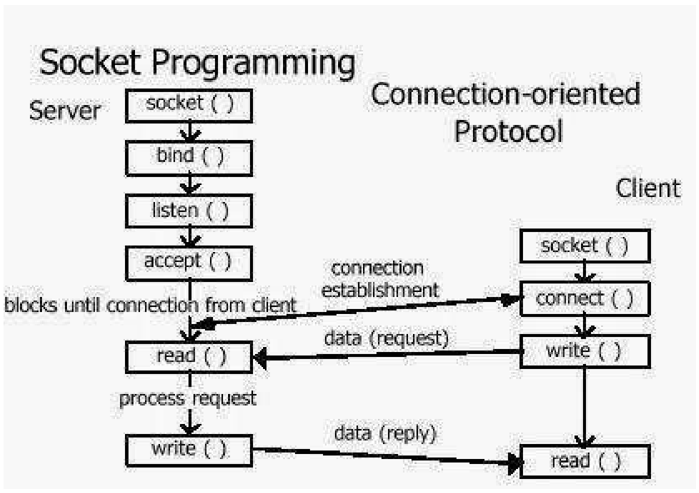
1. IP Address of Server, and
2. Port number.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details.

The java.net package provides support for the two common network protocols,

- **TCP** − TCP stands for Transmission Control Protocol, which allows **reliable, connection oriented** 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 **fast, connection-less protocol** that allows for packets of data to be transmitted between applications.

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. The flow chart of server-client chat is as shown below,



Procedure: *(Ref: 1) https://www.javatpoint.com/socket-programming*
2) https://www.geeksforgeeks.org/socket-programming-in-java/)

Algorithm at Server side
1. Start the program
2. Create a socket in server to client
3. The server establishes a connection to the client.
4. The server accept the connection and to send the data from server to client and vice versa
5. The server communicate the client to send the end of the message
6. Stop the program.

Algorithm at Client Side
1. Start the program
2. Create a socket in client to server.
3. The client establishes a connection to the server.
4. The client accept the connection and to send the data from client to server and vice versa
5. The client communicate the server to send the end of the message
6. Stop the program.

```
//Server side for chat application (save the file as Senddata.java)

import java.io.*;
import java.net.*;
class Senddata{
public static void main(String[] args)throws IOException
{
DatagramSocket ds=new DatagramSocket();
String s="";
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
s=br.readLine();
InetAddress ip=InetAddress.getByName("localhost");
DatagramPacket dp=new DatagramPacket(s.getBytes(),s.length(),ip,8888);
ds.send(dp);
ds.close();
}
}

//Client side for chat application (save the file as Receivedata.java)

import java.io.*;
import java.net.*;
class Receivedata{
public static void main(String[] args)throws Exception
{
DatagramSocket ds=new DatagramSocket(8888);
byte[] b=new byte[1024];
DatagramPacket dp=new DatagramPacket(b,1024);
ds.receive(dp);
String s=new String(dp.getData(),0,dp.getLength());
System.out.println(s);
ds.close();
}
}
```

Open two command prompts. First execute server program and then the client program at respective command prompts. After running the client application, a message will be displayed on the server console.

Implementation:

Implement socket programming in java to read and write on both sides (client and server) using TCP socket.

Output:

Attach print of following as output:

1. Senddata.java and Receivedata.java programs (provided), Program outputs.
2. Client and server tcp read-write socket programs and its outputs.

Post-Experimental Exercise: *(To be written on journal sheet)*

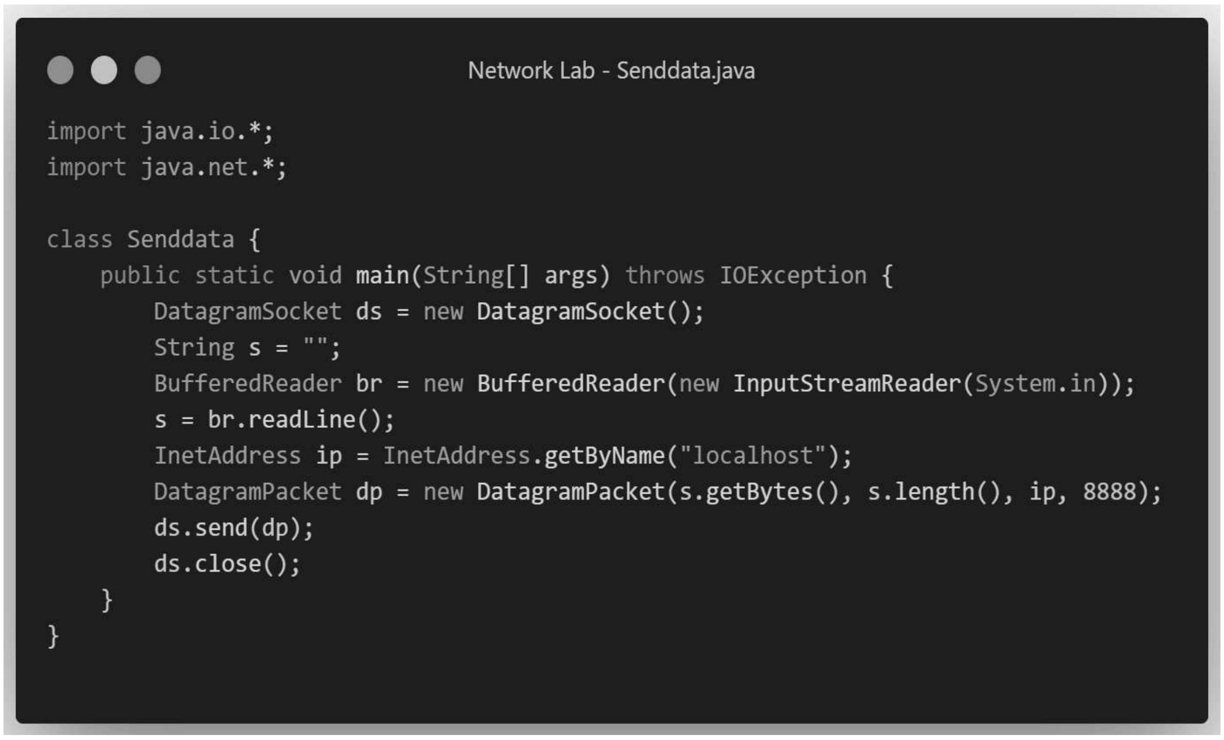
1. Make a table of all important methods with their description that comes under DatagramPacket and DatagramSocket (for UDP) and Socket and ServerSocket (for TCP).

Conclusion: *(To be written on journal sheet)*

Senddata.java and Receivedata.java programs (provided), Program outputs.

Code:

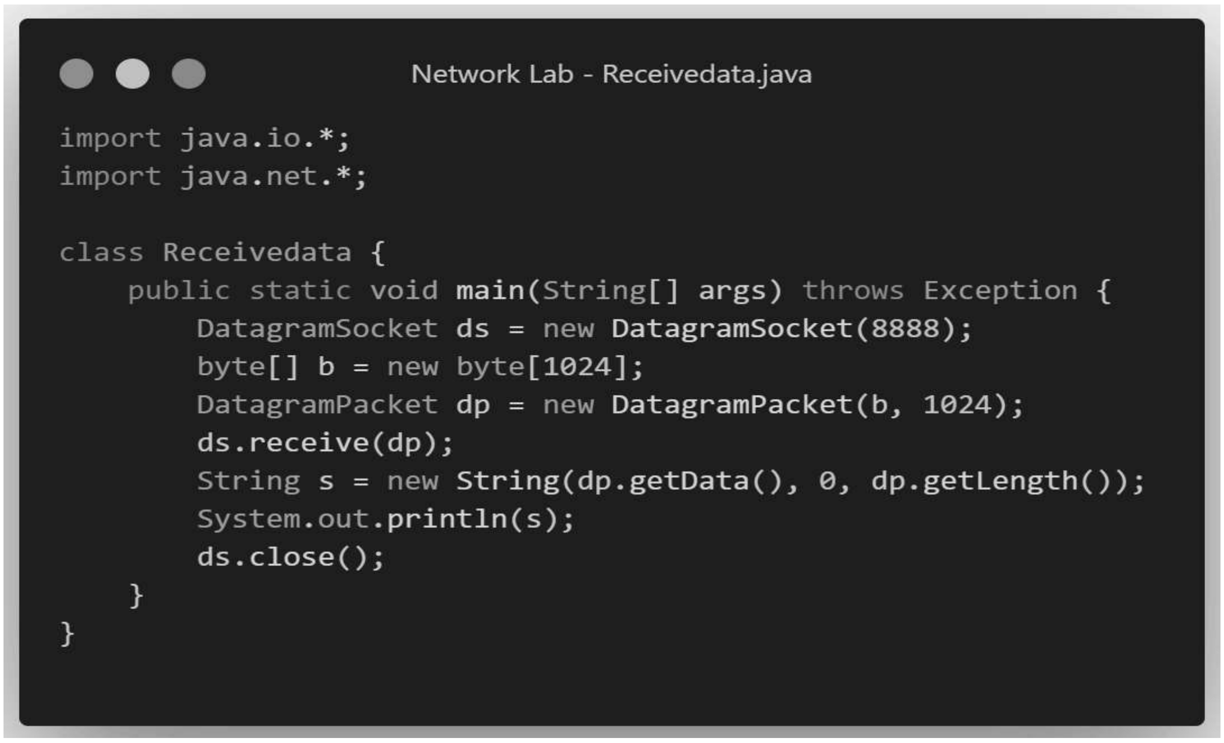
Senddata Code :



Network Lab - Senddata.java

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

class Senddata {
    public static void main(String[] args) throws IOException {
        DatagramSocket ds = new DatagramSocket();
        String s = "";
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        s = br.readLine();
        InetAddress ip = InetAddress.getByName("localhost");
        DatagramPacket dp = new DatagramPacket(s.getBytes(), s.length(), ip, 8888);
        ds.send(dp);
        ds.close();
    }
}
```

Receivedata Code

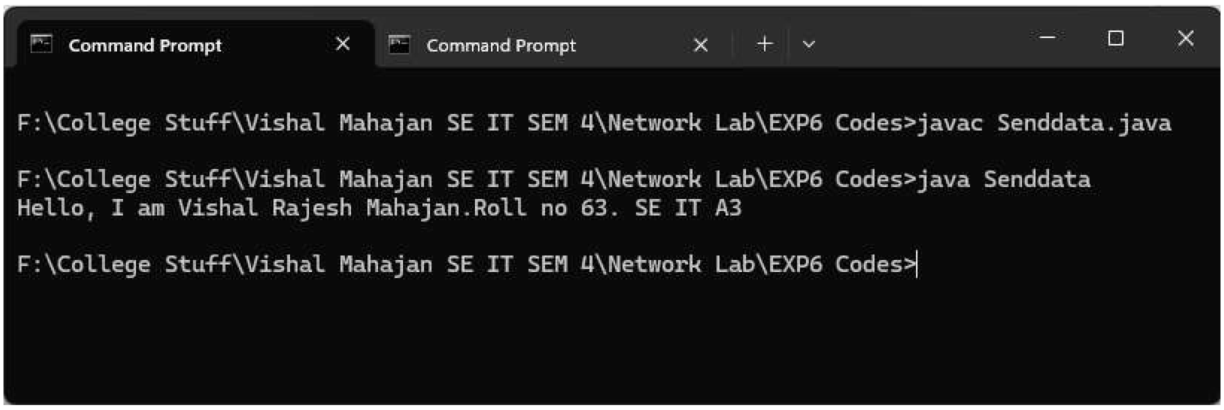


Network Lab - Receivedata.java

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

class Receivedata {
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(8888);
        byte[] b = new byte[1024];
        DatagramPacket dp = new DatagramPacket(b, 1024);
        ds.receive(dp);
        String s = new String(dp.getData(), 0, dp.getLength());
        System.out.println(s);
        ds.close();
    }
}
```

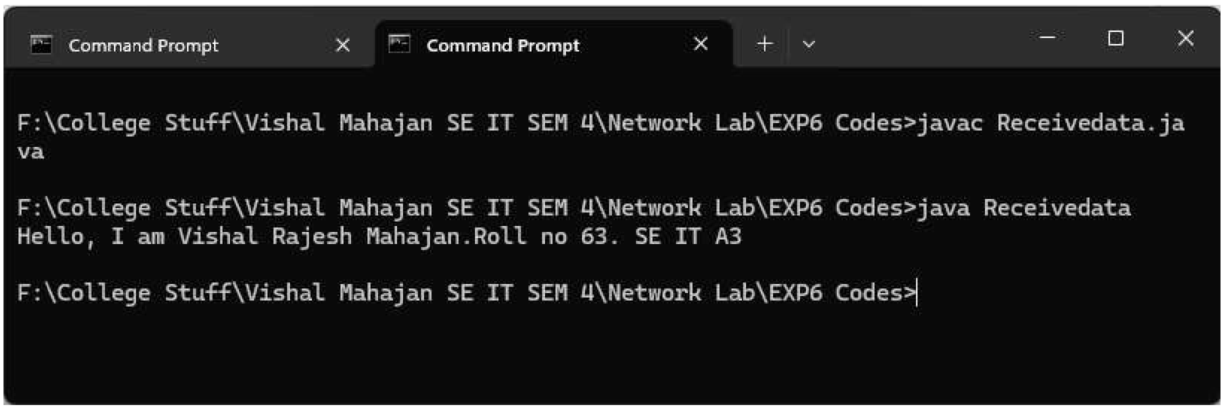
Output:

Send Data Terminal



```
F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>javac Senddata.java

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>java Senddata
Hello, I am Vishal Rajesh Mahajan.Roll no 63. SE IT A3

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>
```

Receive Data Terminal



```
F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>javac Receivedata.ja
va

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>java Receivedata
Hello, I am Vishal Rajesh Mahajan.Roll no 63. SE IT A3

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>
```

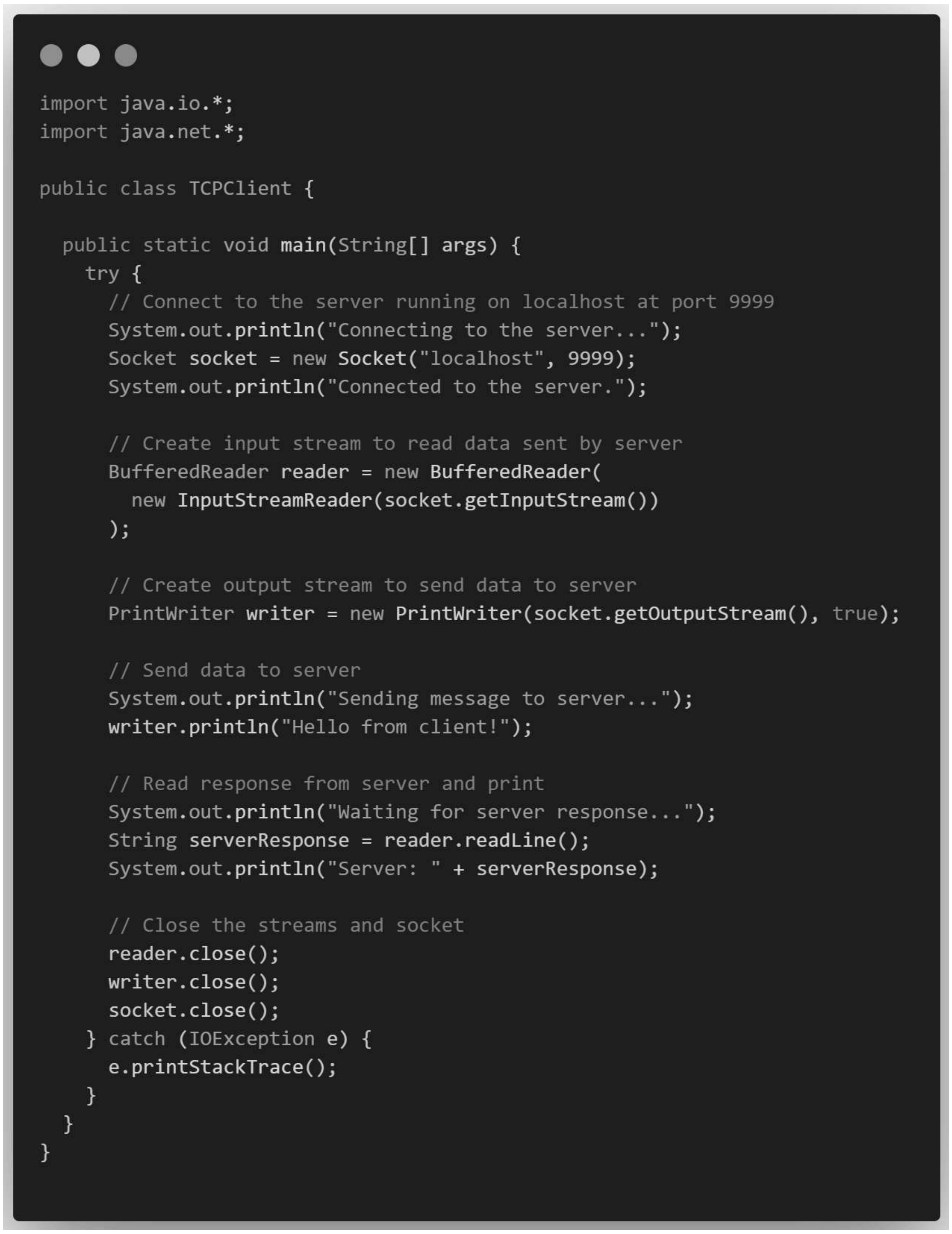
Observation:

The server-side program, Receivedata.java, demonstrates a simple setup where it binds a DatagramSocket to port 8888, awaiting incoming messages from a client. Upon receiving the specific message "Hello, I am Vishal Rajesh Mahajan. Roll no 63. SE IT A3" from the client, it echoes the same message to the console. This program is designed for a single message exchange, as it closes the DatagramSocket after receiving the message.

On the other hand, the client-side program, Senddata.java, prompts the user to input a message. In this instance, the specific message mentioned earlier is hardcoded as the input message. The program then sends this message to the server via a DatagramSocket, supporting only a single message exchange. Interestingly, it does not display any received messages. After sending the message, the DatagramSocket is promptly closed.

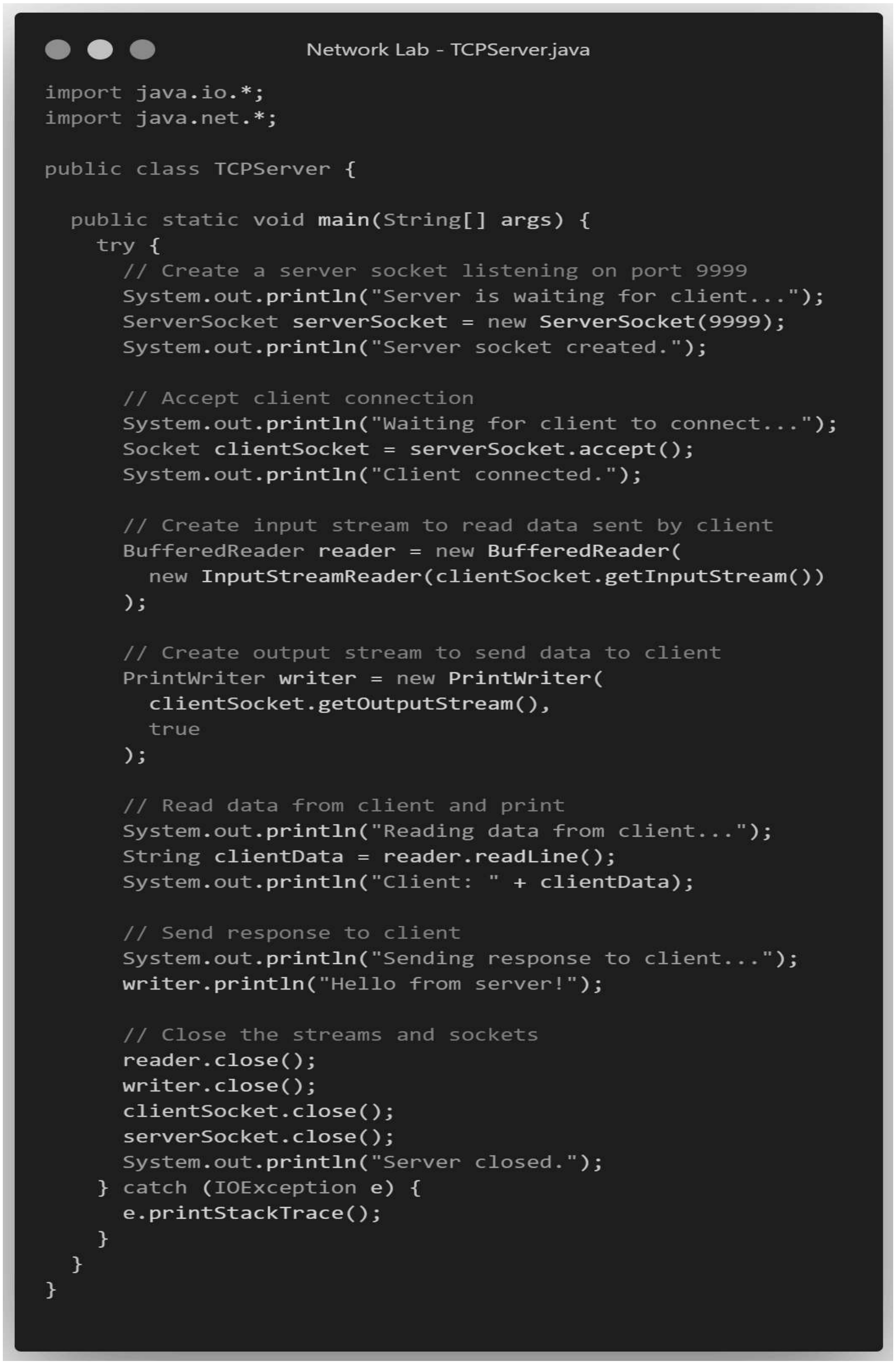
Client and server tcp read-write socket programs and its outputs.

Client Code:



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

public class TCPClient {

  public static void main(String[] args) {
    try {
      // Connect to the server running on localhost at port 9999
      System.out.println("Connecting to the server...");
      Socket socket = new Socket("localhost", 9999);
      System.out.println("Connected to the server.");

      // Create input stream to read data sent by server
      BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream())
      );

      // Create output stream to send data to server
      PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

      // Send data to server
      System.out.println("Sending message to server...");
      writer.println("Hello from client!");

      // Read response from server and print
      System.out.println("Waiting for server response...");
      String serverResponse = reader.readLine();
      System.out.println("Server: " + serverResponse);

      // Close the streams and socket
      reader.close();
      writer.close();
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

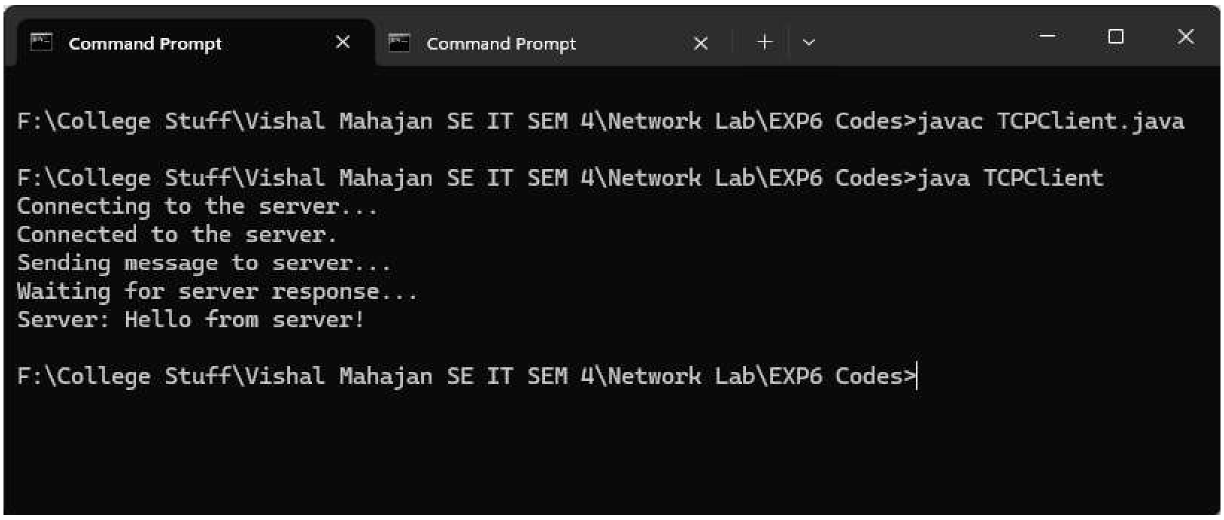
Server Code:



Network Lab - TCPServer.java

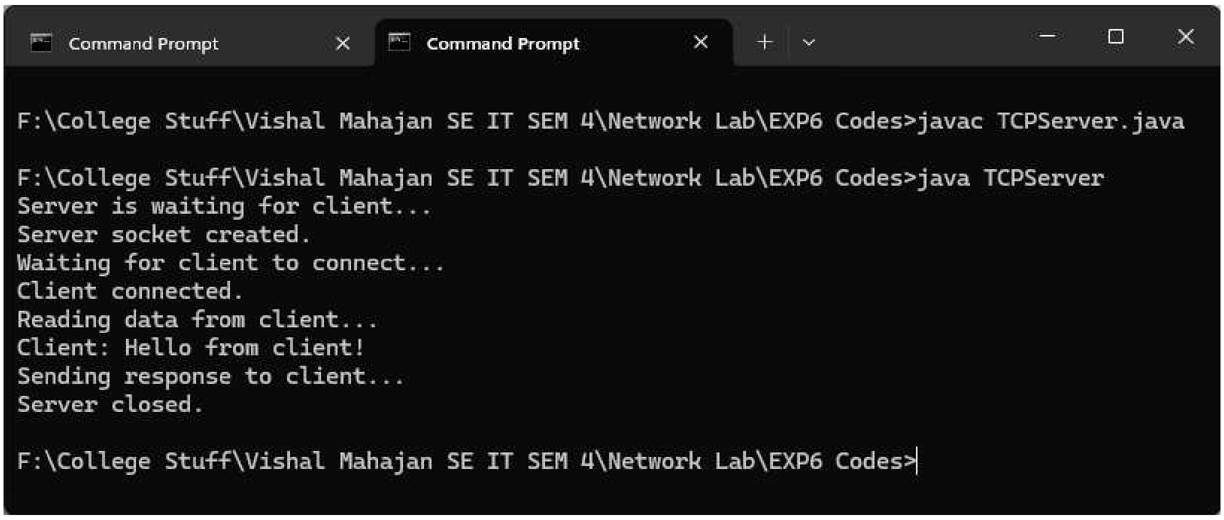
```java
import java.io.*;
import java.net.*;

public class TCPServer {

  public static void main(String[] args) {
    try {
      // Create a server socket listening on port 9999
      System.out.println("Server is waiting for client...");
      ServerSocket serverSocket = new ServerSocket(9999);
      System.out.println("Server socket created.");

      // Accept client connection
      System.out.println("Waiting for client to connect...");
      Socket clientSocket = serverSocket.accept();
      System.out.println("Client connected.");

      // Create input stream to read data sent by client
      BufferedReader reader = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream())
      );

      // Create output stream to send data to client
      PrintWriter writer = new PrintWriter(
        clientSocket.getOutputStream(),
        true
      );

      // Read data from client and print
      System.out.println("Reading data from client...");
      String clientData = reader.readLine();
      System.out.println("Client: " + clientData);

      // Send response to client
      System.out.println("Sending response to client...");
      writer.println("Hello from server!");

      // Close the streams and sockets
      reader.close();
      writer.close();
      clientSocket.close();
      serverSocket.close();
      System.out.println("Server closed.");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Client Output:



```
F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>javac TCPClient.java

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>java TCPClient
Connecting to the server...
Connected to the server.
Sending message to server...
Waiting for server response...
Server: Hello from server!

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>
```

Server Output:



```
F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>javac TCPServer.java

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>java TCPServer
Server is waiting for client...
Server socket created.
Waiting for client to connect...
Client connected.
Reading data from client...
Client: Hello from client!
Sending response to client...
Server closed.

F:\College Stuff\Vishal Mahajan SE IT SEM 4\Network Lab\EXP6 Codes>
```

Observation:The **TCPClient.java** program creates a socket to connect to a server running on localhost at port 9999. It sets up input and output streams for communication with the server. The client sends a message to the server, waits for a response, and then prints the response message. After the communication is complete, it closes the streams and the socket to release resources. The **TCPServer.java** program initializes a ServerSocket on port 9999 and waits for incoming client connections. Once a connection is established, it sets up input and output streams to communicate with the client. After receiving a message from the client, it prints the message and sends a response back. Finally, it closes the streams and sockets to release resources.