VIA University College

Software Development with UML and Java 2

Joseph Okika (jook@via.dk)

Learning Objectives

- Sockets
  - Know how to find the host machine's IP address through a Java program
  - Write small programs that use TCP sockets in both client and server programs.
  - Write small programs that use UDP sockets in both client and server programs
  - Understand the convenience of Java's stream classes and use of JSON

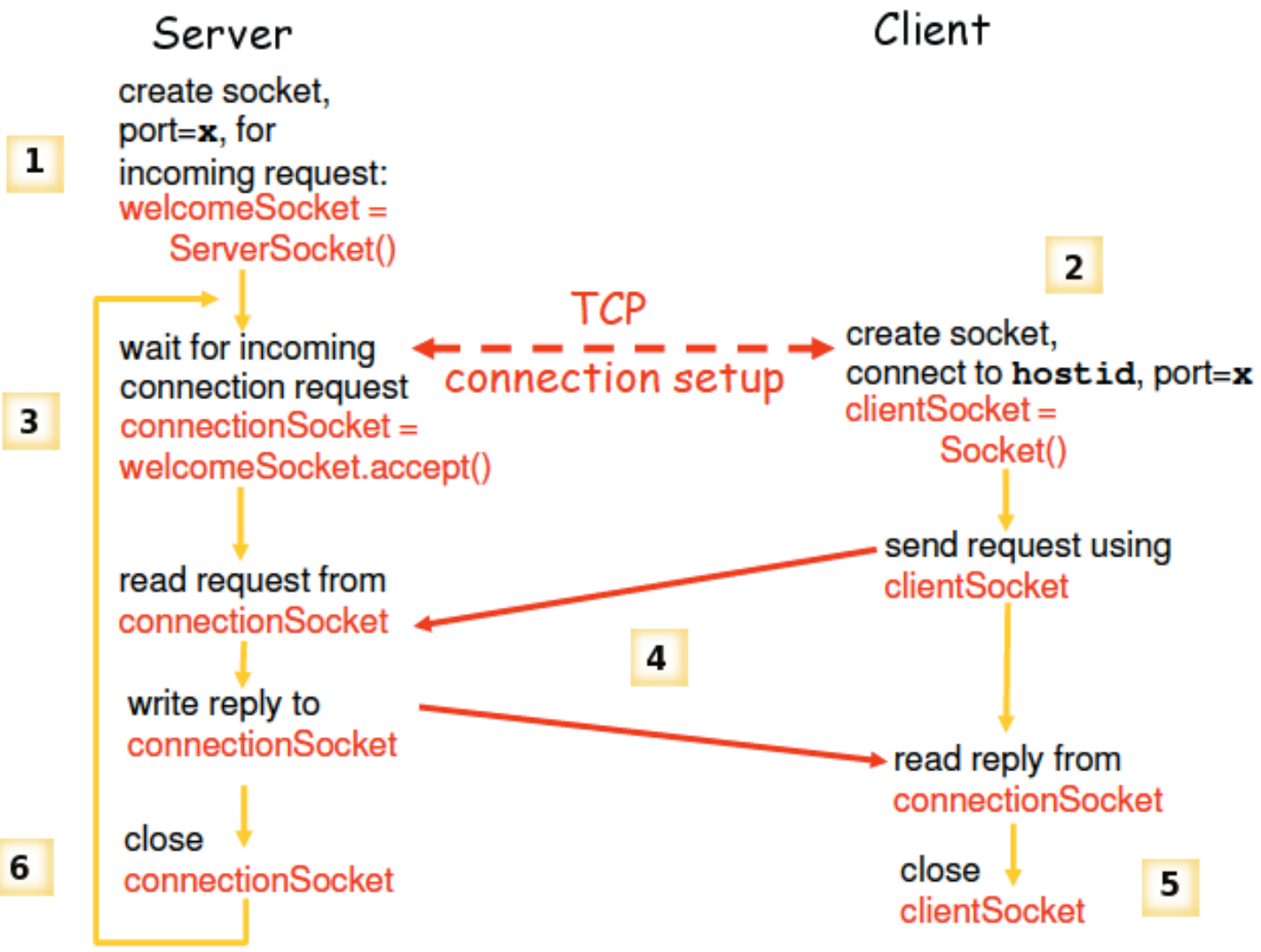
What is a socket?

- a stream that connects two processes running in different address spaces (usually across a network or on the same machine)
- provides an interface to send data to/from the network through a port
- creating a socket between two machines can be seen as creating an input and output streams for sending data between programs running on each machine
- lowest-level form of communication from application developer's view.
- Higher-level techniques
  - Message passing systems(MPI, SOAP, JMS)
  - Web servers extensions(ASP, JSP, servlets)
  - Distributed objects (CORBA, RMI), web services

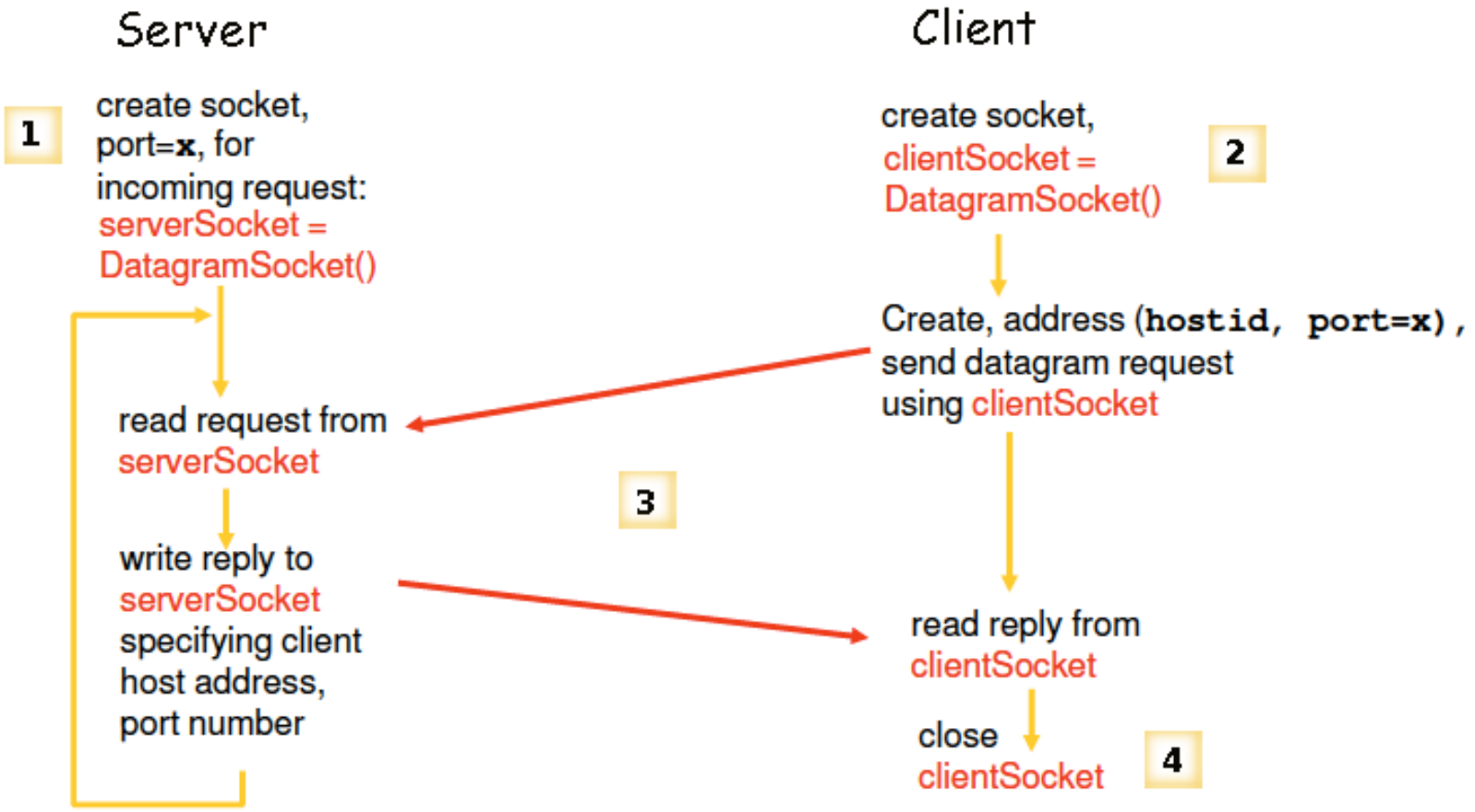
Basic Client-Server Program

- Client: write a program that rings up another program at a specified IP address and running on a specified port.
- Server: write a second program that accepts connection and establishes input/output stream to client.
- When server accepts connection, client can establish input/output stream to server.
- Client can now make request by sending data. Server sends replies to client.

TCP Sockets


Server
Client
create socket, port=x, for incoming request: welcomeSocket = ServerSocket()
1
wait for incoming connection request connectionSocket = welcomeSocket.accept()
3
TCP connection setup
2
create socket, connect to hostid, port=x clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
4
write reply to connectionSocket
read reply from connectionSocket
close clientSocket
5
close connectionSocket
6

UDP Sockets


Server
Client
1
create socket, port=x, for incoming request: serverSocket = DatagramSocket()
create socket, clientSocket = DatagramSocket()
2
Create, address (hostid, port=x), send datagram request using clientSocket
read request from serverSocket
3
write reply to serverSocket specifying client host address, port number
read reply from clientSocket
close clientSocket
4

TCP Socket vs UDP Socket

- Stream Socket:
  - A dedicated point-to-point channel between a client and server.
  - Use TCP (Transmission Control Protocol) for data transmission.
  - Lossless and reliable.
  - Sent and received in the same order.
- Datagram Socket:
  - No dedicated point-to-point channel between a client and server.
  - Use UDP (User Datagram Protocol) for data transmission.
  - May lose data and not 100% reliable.
  - Data may not be received in the same order as sent.

Socket Programming – Java classes

- TCP
  - java.net.Socket
  - Java.net.ServerSocket
  - java.net.InetAddress
- UDP…
  - java.net.DatagramPacket
  - java.net.DatagramSocket

Socket programming – classes/methods

- java.net.Socket
  - Socket(InetAddress *addr*, int *port*);
    - create a Socket connection to address *addr* on port *port*
  - InputStream getInputStream();
    - returns an instance of InputStream for getting info from the implicit Socket object
  - OutputStream getOutputStream();
    - returns an instance of OutputStream for sending info to implicit Socket object.
  - close();
    - close connection to implicit socket object, cleaning up resources
- java.net.ServerSocket
- java.net.InetAddress

Socket Programming – classes/methods

- java.net.ServerSocket
  - ServerSocket(int *port*);
    - enables program to listen for connections on port *port*
  - Socket accept();
    - blocks until connection is requested via Socket request from some other process. When connection is established, an instance of Socket is returned for establishing communication streams.

Socket Programming – classes/methods

- java.net.InetAddress
  - static InetAddress getByName(String *name*)
    - given a hostname *name*, return the InetAddress object representing that name (basically encapsulates name and IP associated with name);
  - static InetAddress getLocalHost()
    - get InetAddress object associated with local host.
  - static InetAddress getByAddress(byte[] *addr*)
    - get InetAddress object associated with address *addr*

Example: Reading from & Writing to a Socket

- Implements a client TCPClient, that connects to the server
- The server, TCPServer receives messages from its client keeping track of the number of messages and sending back each message with together with the message count.

TCP Client

- Creates a socket to establish a connection to the server
- Reads input from the user on the standard input stream.
- Sends the input text to the server using the socket.
- Reads the message passed back from the server (after the server sends the message through the socket)

TCPClient Code

```
import java.io.*;
import java.net.*;
import java.util.*;
/*
 * sends messages or "Exit" to close down the connection.
 */
public class TCPClient {
  private static InetAddress host;
  private static final int PORT = 3456;

  public static void main(String[] args) {
    try {
      host = InetAddress.getLocalHost();
    } catch (UnknownHostException uhe) {
      System.out.println("Unable to find the Host ID!");
      System.exit(1);
    }
    accessServer();
  }
```

TCPClient code conti.

```
    private static void accessServer() {
    Socket clientSocket = null;
    try {
       // Step 1: create client socket and establish a connection to the server
       clientSocket = new Socket(host, PORT);
       // Step 2: Setup input and output streams
       DataInputStream inFromServer =
                        new DataInputStream(clientSocket.getInputStream());
       DataOutputStream outToServer =
                        new DataOutputStream(clientSocket.getOutputStream());
       // create keyboard input stream
       Scanner userInput = new Scanner(System.in);
       String msg, resp;
       do {
          // read line from user input
          System.out.println("Enter a message or (enter \"Exit\" to close): ");
          msg = userInput.nextLine();
          // Step 3: Send and receive data
          outToServer.writeUTF(msg);
          resp = inFromServer.readUTF();
          System.out.println("\nFROM SERVER> " + resp);
       } while (!msg.equals("Exit"));
    } catch (IOException ioe) {
    } catch (NoSuchElementException nse) {
    } finally {
       try {
          System.out.println("\n Closing the connection to the server!");
          // Step 4: Close the connection
          clientSocket.close();
       } catch (IOException ioe) {
          System.out.println("Unable to close te connection t o the server!");
          System.exit(1);
       }
    }
  }
}
```

TCPClient Java code

```
InetAddress host = InetAddress.getLocalHost();
clientSocket = new Socket(host, PORT);
```

- Initiate a TCP connection with the host through port 3456

```
DataInputStream inFromServer =
                new DataInputStream(clientSocket.getInputStream());
DataOutputStream outToServer =
                new DataOutputStream(clientSocket.getOutputStream());
```

- Creates input and output stream to handle input/output to server

```
outToServer.writeUTF(msg);
resp = inFromServer.readUTF();
```

- Send and receive message to/from server

```
clientSocket.close();
```

- Close the connection

TCPServer Code

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

public class TCPServer {
   private static ServerSocket welcomingSocket;
   private static final int PORT = 3456;
   private static String clientIP;

   public static void main(String[] args) {
      System.out.println("Starting Server...");
      // Step 1: Creating the server welcoming socket
      try {
         welcomingSocket = new ServerSocket(PORT);
      } catch (IOException ioe) {
         System.out.println("Unable to connect with the given port!");
         System.exit(1);
      }
      do
      {
         System.out.println("Waiting for a client...");
         handleClient();
      } while(true);
   }
```

TCPServer Code conti.

```
    private static void handleClient() {
    Socket connSocket = null;
    try {
       // Step 2: put the server into a waiting state for contact by the client
       connSocket = welcomingSocket.accept();

       clientIP = connSocket.getInetAddress().getHostAddress();
          System.out.println("Welcome " + clientIP);

       // Step 3 : setup input and output streams
       DataInputStream inFromClient = new DataInputStream(connSocket.getInputStream());
       DataOutputStream outToClient = new DataOutputStream(connSocket.getOutputStream());

       int msgCount = 0;

       // Step 4: Send and receive data
       String message = inFromClient.readUTF();
       while(!message.equals("Exit"))
       {
          System.out.println("Received Message. OK!");
          msgCount++;
          outToClient.writeUTF("Message number " + msgCount + ": "+ message);
          message = inFromClient.readUTF();
       }
    } catch (IOException ioe) {
       ioe.printStackTrace();
    }

       try
       {
          System.out.println("\n Now closing connection...");
          // Step 5: Close the connection
          connSocket.close();
       } catch (IOException e) {
          System.out.println("Unable to close the connection!");
          System.exit(1);
       }
    }
}
```

TCPServer Java code

```
ServerSocket welcomingSocket = new ServerSocket(PORT);
```

- Creates a welcoming socket that handles client connection from port 3456

```
Socket connSocket  =  welcomingSocket.accept();
```

- Creates a new socket and goes into a waiting state

UDPClient

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

public class UDPClient
{
   public static void main(String args[]) throws IOException
   {
      final int PORT = 9876;
      final String HOST = "localhost";
      // create input stream
      BufferedReader inFromUser = new BufferedReader(
                                  new InputStreamReader(System.in));

      // create client socket
      DatagramSocket clientSocket = new DatagramSocket();

      // Translate hostname to IP using DNS
      InetAddress IPAddress = InetAddress.getByName(HOST);
      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];
```

UDPClient conti.

```
        // Read input from user
        System.out.print("Write a line for the server: ");
        String sentence = inFromUser.readLine();
        System.out.println("Client> " + sentence);
        sendData = sentence.getBytes();

        // Create datagram with data-to-send, length, IP addr, port
        DatagramPacket sendPacket = new DatagramPacket(sendData,
                                        sendData.length,IPAddress, PORT);

        // Send datagram to server
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
                                                    receiveData.length);

        // Read datagram from server.
        clientSocket.receive(receivePacket);
        String modifiedStc = new String(receivePacket.getData()).trim();
        System.out.println("Server> " + modifiedStc);

        // Close connection.
        clientSocket.close();
    }
}
```

UDPServer

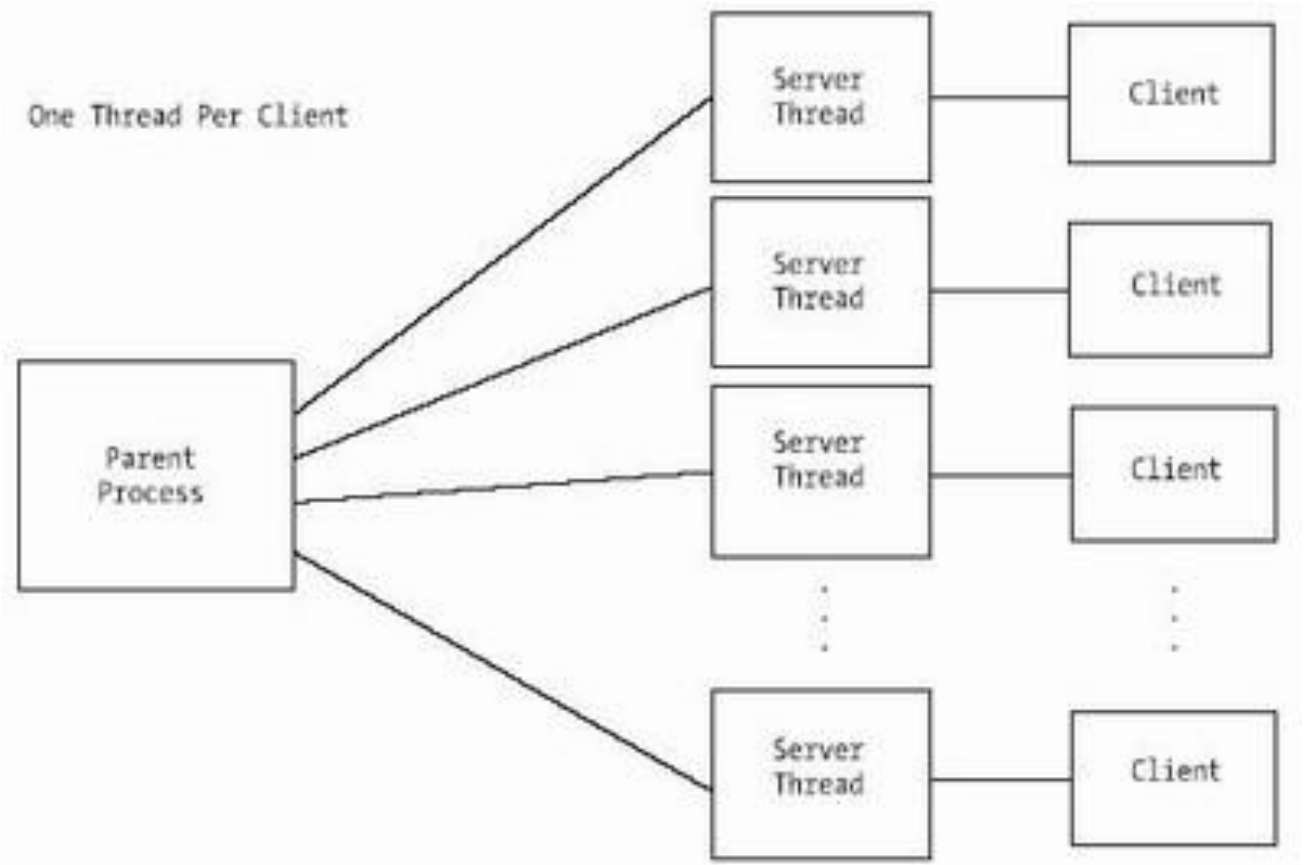
```
import java.io.*; import java.net.*;
public class UDPServer
{
   public static void main(String args[]) throws IOException
   {  final int PORT = 9876;
      System.out.println("Starting Server...");
      // Create UDP server socket at port 9876.
      DatagramSocket serverSocket = new DatagramSocket(PORT);
      while (true)
      {
         System.out.println("Waiting for a client...");
         // Create space for receiving datagram
         byte[] receiveData = new byte[1024];
         DatagramPacket receivePacket = new DatagramPacket(receiveData,
               receiveData.length);
         // Receive datagram from client
         serverSocket.receive(receivePacket);
         String sentence = new String(receivePacket.getData()).trim();
```

UDPServer conti.

```
            // Get IP addr and port number of the client
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();

            System.out.println("Client> " + sentence);
            String capitalizedSentence = sentence.toUpperCase();
            System.out.println("Server> " + capitalizedSentence);

            byte[] sendData = new byte[1024];
            sendData = capitalizedSentence.getBytes();

            // create datagram to send to the client.
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                  sendData.length, IPAddress, port);

            // Write out datagram to socket.
            serverSocket.send(sendPacket);

            // loop back and wait for another client connection
        }
    }
}
```

Multi-Threaded Server

- The server should not block for other clients while communicating with a client



Multi-threaded TCPServer

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

public class ThreadedTCPServer {
  private static ServerSocket listeningSocket;
  private static final int PORT = 3456;

  // create server socket listening at the given port.
  public ThreadedTCPServer() {
    try {
      // Step 1: Creating the server welcoming socket
      listeningSocket = new ServerSocket(PORT);
      System.out.println("ServerSocket: " +
listeningSocket);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Multi-threaded TCPServer II

```
  private void listenToClient() {
    Socket connSocket = null;
    while (true) { // run until you terminate the program
            try {
               // Block until a connection is made.
              connSocket = listeningSocket.accept();
        System.out.println("Socket: " + connSocket);
               // Start a new thread for each client
        Thread ct = new Thread(new
                        TCPClientThreadHandler(connSocket));
        ct.start();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
  }

  public static void main(String[] args) {
        new ThreadedTCPServer().listenToClient();
     }
}
```

TCPClient Thread Handler

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

public class TCPClientThreadHandler implements Runnable {
  private Socket socket;
  private DataInputStream inFromClient;
  private DataOutputStream outToClient;

  public TCPClientThreadHandler(Socket socket) {
    this.socket = socket;
  }

  @Override
  public void run() {
    try {
      // create input stream attached to the socket.
      inFromClient = new
                    DataInputStream(socket.getInputStream());

      // create output stream attached to the socket.
      outToClient = new
                   DataOutputStream(socket.getOutputStream());
```

TCPClient Thread Handler II

```
      int msgCount = 0;
      // Step 4: Send and receive data
      String message = inFromClient.readUTF();
      while (!message.equals("Exit")) {
        System.out.println("Received Message. OK!");
        msgCount++;
        outToClient.writeUTF("Message number " + msgCount +
                                                ": " + message);
        message = inFromClient.readUTF();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
    try {
      System.out.println("\n Now closing connection..." +
                                              " with " + socket );
      // Step 5: Close the connection
      socket.close();
    } catch (IOException e) {
      System.out.println("Unable to close the connection!");
      System.exit(1);
    }
  }
}
```