Bộ môn An toàn Thông tin – Khoa MMT&TT
Trường Đại học Công nghệ Thông tin (UIT)

LAB REPORT

Subject: Basic network programming

(Session 02)

TOPIC: Lab 3

Group: Ningguang

1. GENERAl INFORMATION:

Class: NT106.M21.ATCL.1

Ordinal number	Full name	Student ID	Email
1	Truong Thi Hoang Hao	20520191	20520191@gm.uit.edu.vn
2	Nguyen Dinh Kha	20520562	20520562@gm.uit.edu.vn

2. IMPLEMENTATION CONTENT:

Ordinal number	Work	Self – assessment result
1	Scenario 01 – UDP Client & Server	100%
2	Scenario 02 – Telnet & TCP Server	100%
3	Scenario 03 – TCP Client & Listener	100%
4	Scenario 04 – Server & Multiclient	100%

The section below this report is the detailed report document of the implementation team

DETAILED REPORT

1. Scenario 01

- Resource:
- Objective:
  To create an application that can send and receive data between two parties (a client and a server) using the UDP protocol. The client user specifies the IP address, port to connect to, and the message to send to the server. The server receives the message sent from the client.

- **Steps to implement:**

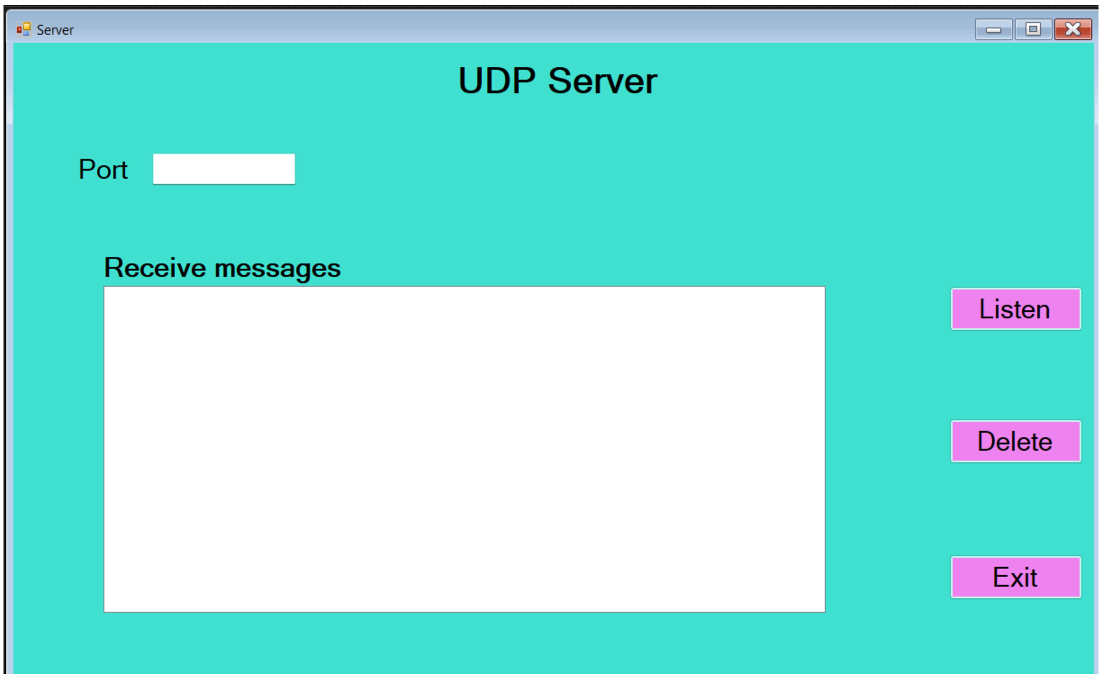
– **Step 1: Design the Interface**

– UDP Server Interface:

+ A textbox (txt_Port) for entering the Port.

+ A listview to display messages received from the client.

+ Three buttons: Listen (to receive messages from the client), Delete (to clear all entered data for new inputs), and Exit (to close the program).



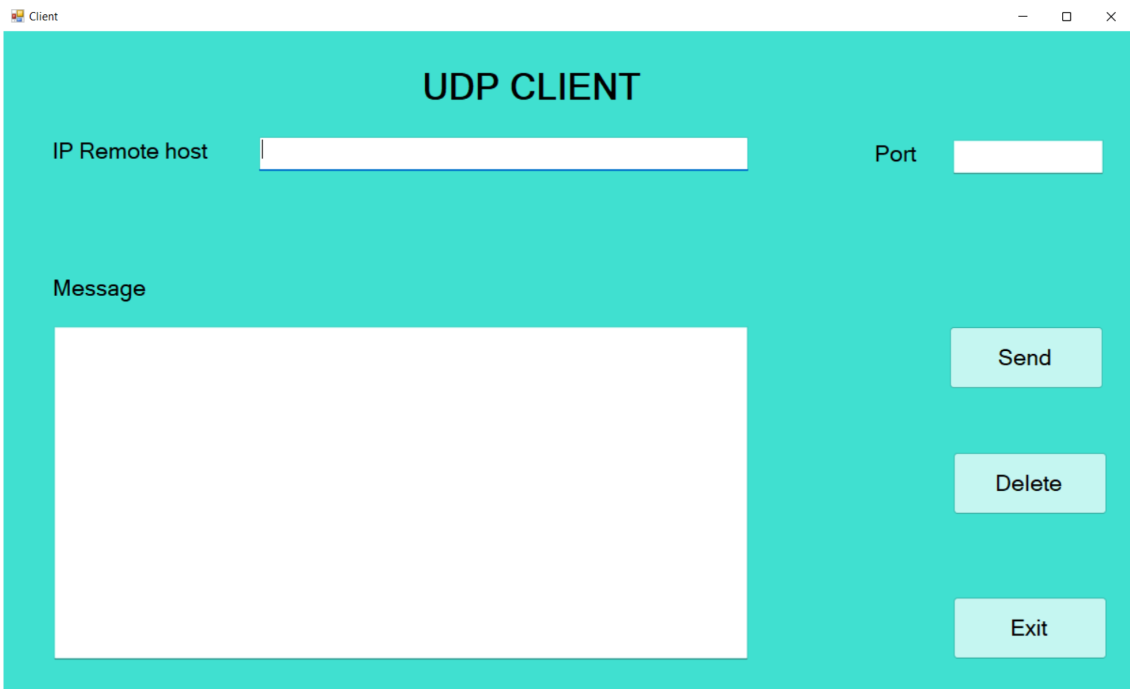
- UDP Client Interface:

+ A textbox for the IP address of the remote host.

+ A textbox for the port number.

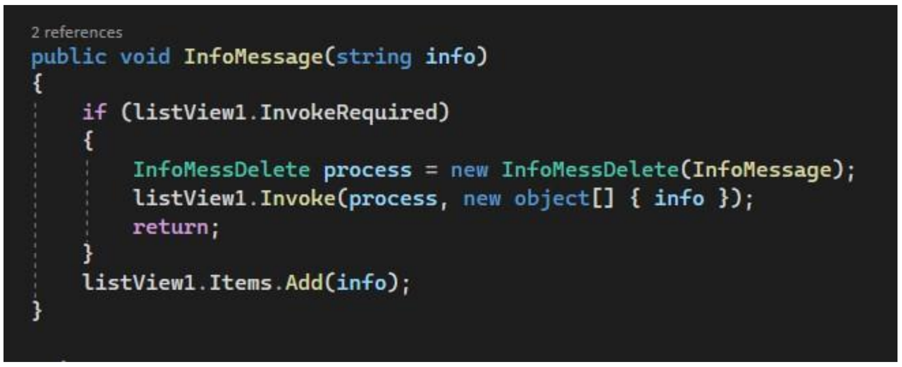
+ A textbox for entering the message to be sent.

+ Buttons for Send, Delete (clear data), and Exit (close the application).



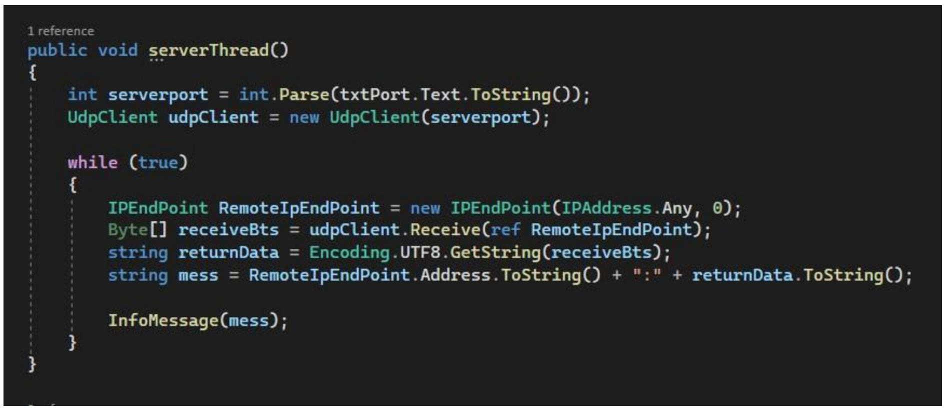
– **Step 2: Proceed with Coding**

+ UDP Server:



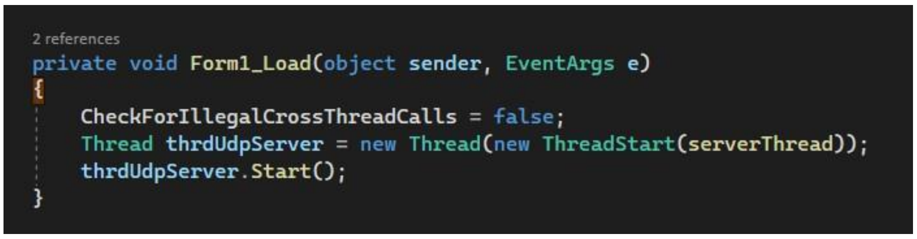
```
2 references
public void InfoMessage(string info)
{
    if (listView1.InvokeRequired)
    {
        InfoMessDelete process = new InfoMessDelete(InfoMessage);
        listView1.Invoke(process, new object[] { info });
        return;
    }
    listView1.Items.Add(info);
}
```

▪ InfoMessage function to display messages on the listView.



```
1 reference
public void serverThread()
{
    int serverport = int.Parse(txtPort.Text.ToString());
    UdpClient udpClient = new UdpClient(serverport);

    while (true)
    {
        IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any, 0);
        Byte[] receiveBts = udpClient.Receive(ref RemoteIpEndPoint);
        string returnData = Encoding.UTF8.GetString(receiveBts);
        string mess = RemoteIpEndPoint.Address.ToString() + ":" + returnData.ToString();

        InfoMessage(mess);
    }
}
```

- serverThread function to receive data sent from the client and display it on the listView. This involves creating a UDP connection and using a variable (serverport) to store the port number entered in the txtPort textbox.

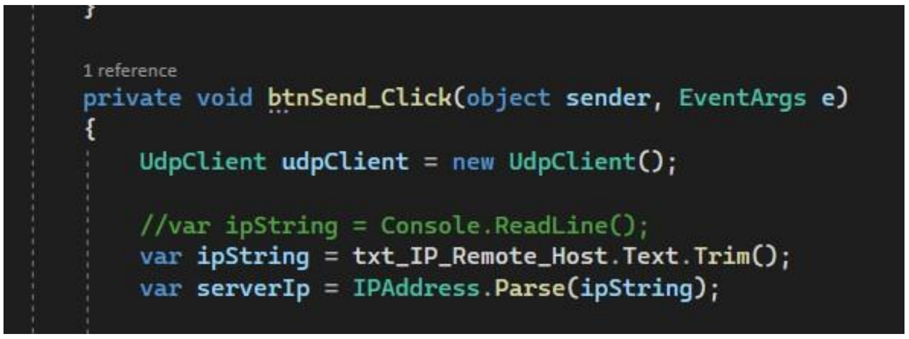


```
2 references
private void Form1_Load(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Thread thrdUdpServer = new Thread(new ThreadStart(serverThread));
    thrdUdpServer.Start();
}
```

- On the Form1_Load event, create a thread to run the function that receives data from the client.

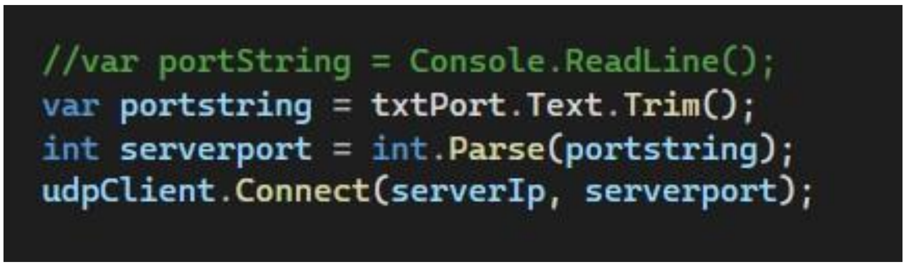
+ UDP Client:

- Send button functionality to send a message to the server..



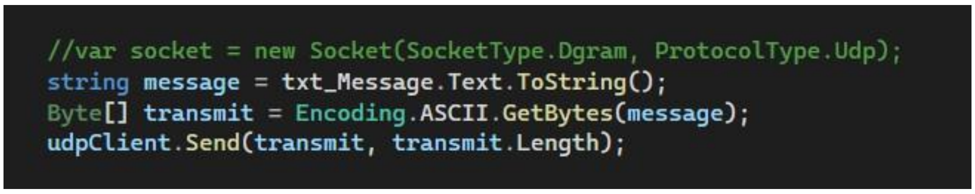
```
    }

    1 reference
    private void btnSend_Click(object sender, EventArgs e)
    {
        UdpClient udpClient = new UdpClient();

        //var ipString = Console.ReadLine();
        var ipString = txt_IP_Remote_Host.Text.Trim();
        var serverIp = IPAddress.Parse(ipString);
```

- Initialize variables for IP address (ipString, serverIp) and port number (portString, serverport) based on user input.



```
//var portString = Console.ReadLine();
var portstring = txtPort.Text.Trim();
int serverport = int.Parse(portstring);
udpClient.Connect(serverIp, serverport);
```

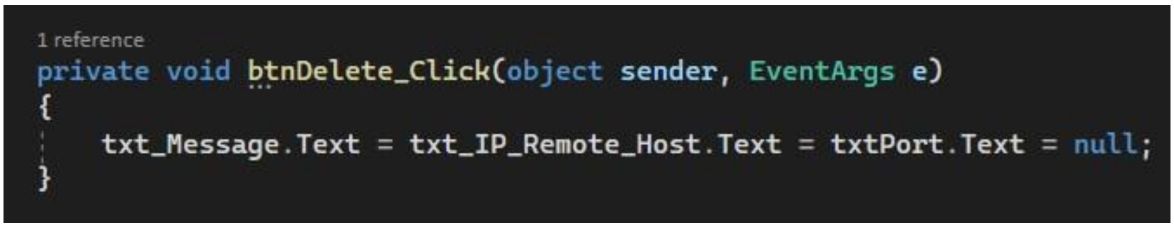
- Declare a message string to store the input from the txt_Message textbox, convert this message to bytes, and store it in a variable (transmit). Use UdpClient.Send to send the UDP data to the Remote Host.



```
//var socket = new Socket(SocketType.Dgram, ProtocolType.Udp);
string message = txt_Message.Text.ToString();
Byte[] transmit = Encoding.ASCII.GetBytes(message);
udpClient.Send(transmit, transmit.Length);
```

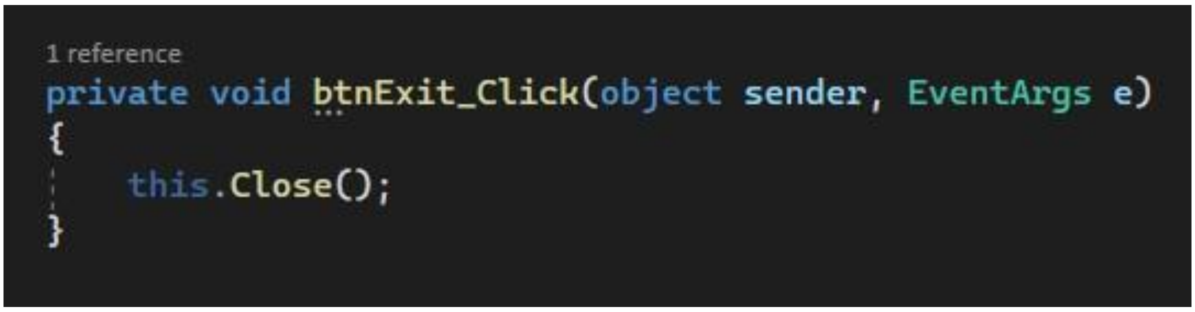
- Declare a string variable to store the message entered into the textbox txt_Message.Text.
- Convert the message string to a byte array
- Use the UdpClient.Send method to send the UDP data to the remote host

- Delete button to clear all input data.



```
1 reference
private void btnDelete_Click(object sender, EventArgs e)
{
    txt_Message.Text = txt_IP_Remote_Host.Text = txtPort.Text = null;
}
```

- Exit button to close the form and terminate the program.



```
1 reference
private void btnExit_Click(object sender, EventArgs e)
{
    this.Close();
}
```

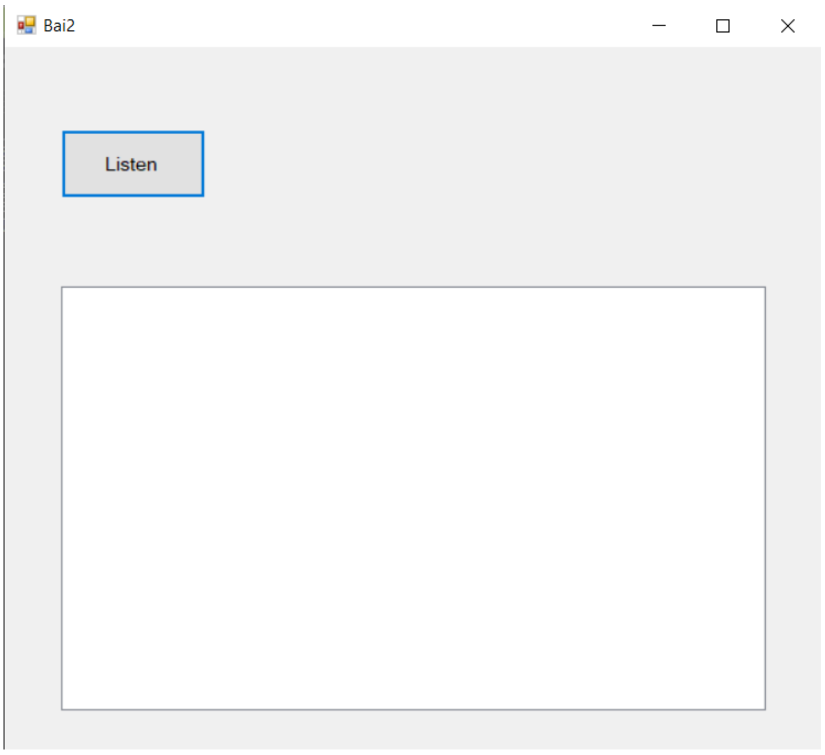
2. Scenario 2

- **Resource:** Attached compressed file

- **Objective:** Write a program to listen for data from the Telnet service using TCP connection (using the Socket class) with the following description:
  - Run the program.
  - Click the Listen button.
  - Open CMD (Command Prompt) and type the command: telnet <Your machine's IP> 8080
  - On the telnet screen, type any message, and the program will receive it and display it on the form.
  - See the sample image.

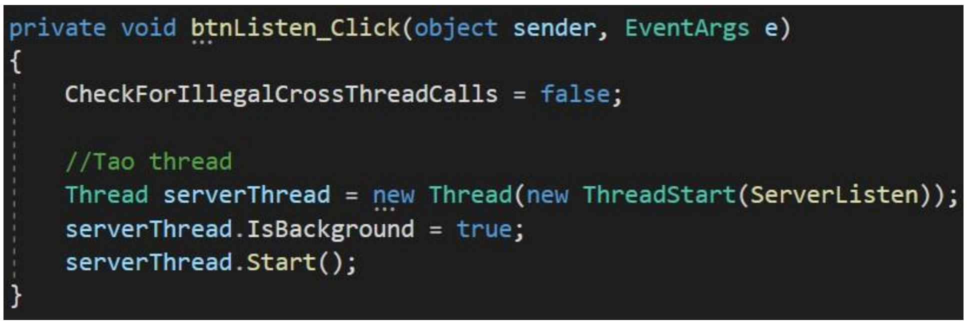
- **Steps to implement:**

Step 1: Design the interface which includes a 'Listen' button for the server to start listening and a listview to display information.



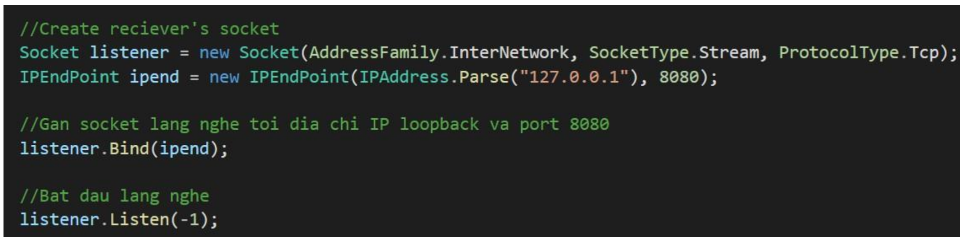
Step 2: Code the functionality of the 'Listen' button

Initially, create a thread and then create a ServerListen function for the thread.



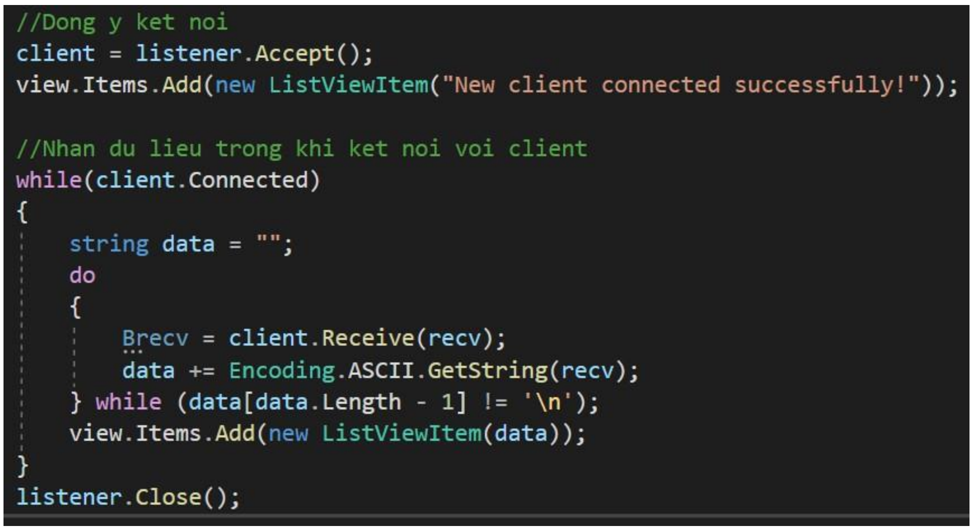
```
private void btnListen_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;

    //Tao thread
    Thread serverThread = new Thread(new ThreadStart(ServerListen));
    serverThread.IsBackground = true;
    serverThread.Start();
}
```

In the ServerListen function, create a receiving socket and set this socket to listen for incoming connections at the loopback IP address and port 8080.



```
//Create reciever's socket
Socket listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
IPEndPoint ipend = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8080);

//Gan socket lang nghe toi dia chi IP loopback va port 8080
listener.Bind(ipend);

//Bat dau lang nghe
listener.Listen(-1);
```

Next, the sending socket agrees to connect with the receiving socket. If the connection is successful, the listview will display the message “New client connected successfully!” and will begin to receive data and display it in the listview for the duration of the connection. Finally, the receiving socket will close the connection.



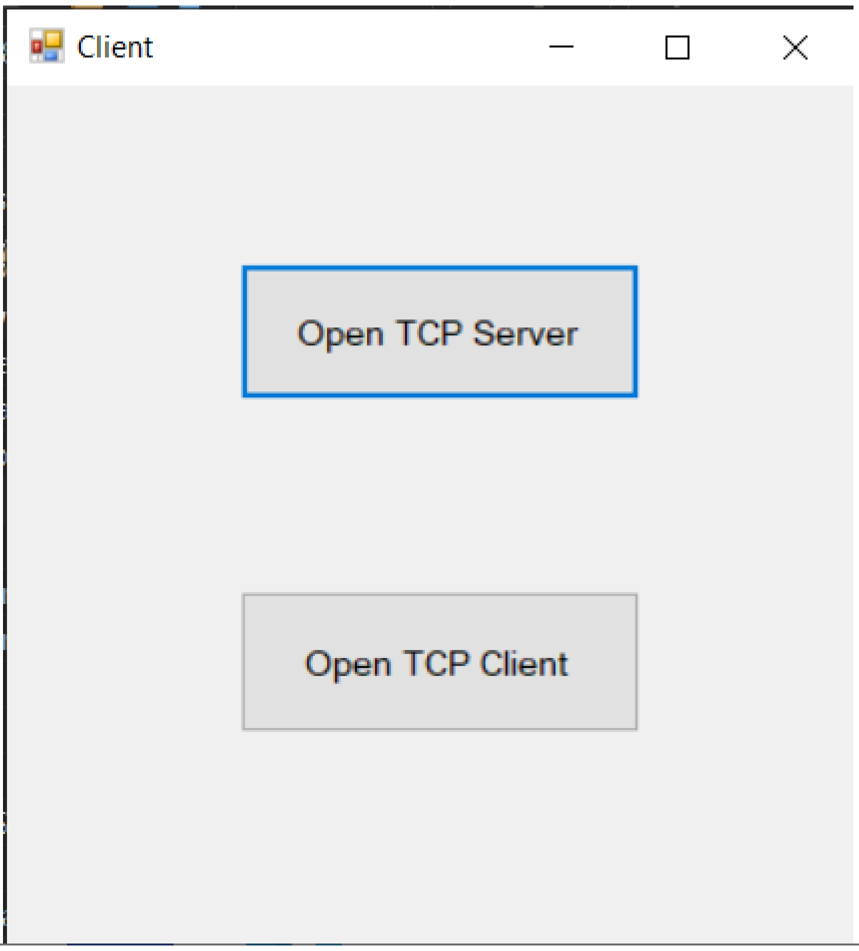
```
//Dong y ket noi
client = listener.Accept();
view.Items.Add(new ListViewItem("New client connected successfully!"));

//Nhan du lieu trong khi ket noi voi client
while(client.Connected)
{
    string data = "";
    do
    {
        Brecv = client.Receive(recv);
        data += Encoding.ASCII.GetString(recv);
    } while (data[data.Length - 1] != '\n');
    view.Items.Add(new ListViewItem(data));
}
listener.Close();
```

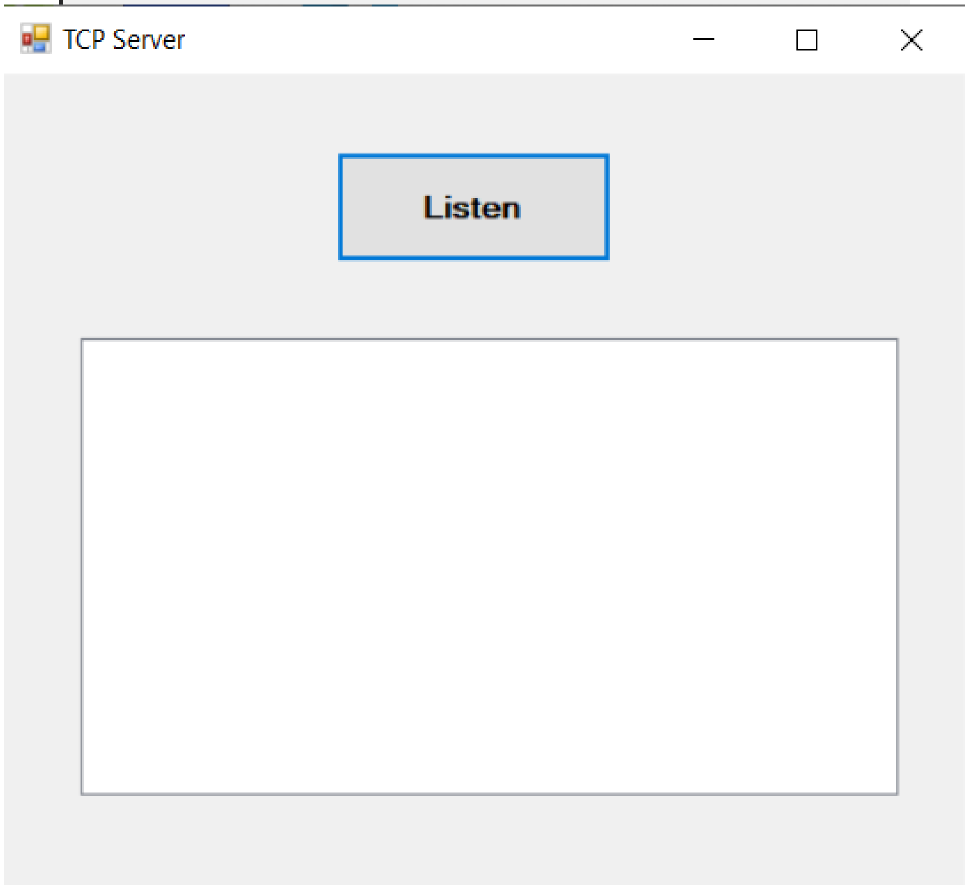
3. Scenario 03

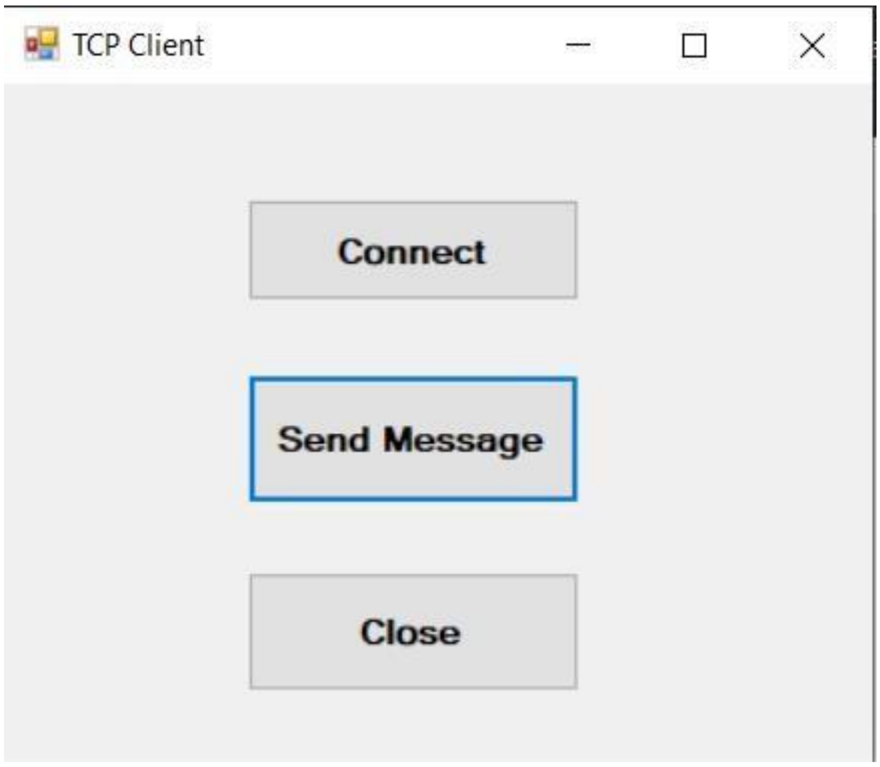
- **Resource:**
- **Objective:** Write an application to send and receive data using TCP protocol (TCP Client and TCP Listener). The server listens for connection and message from Client.
  - Run the Server.
  - Click the 'Listen' button.
  - Initialize the Client.
  - Send the message from Client to Server.
  - The server receives the message and displays it on the form.
  - View the sample image.

- **Steps to implement:**

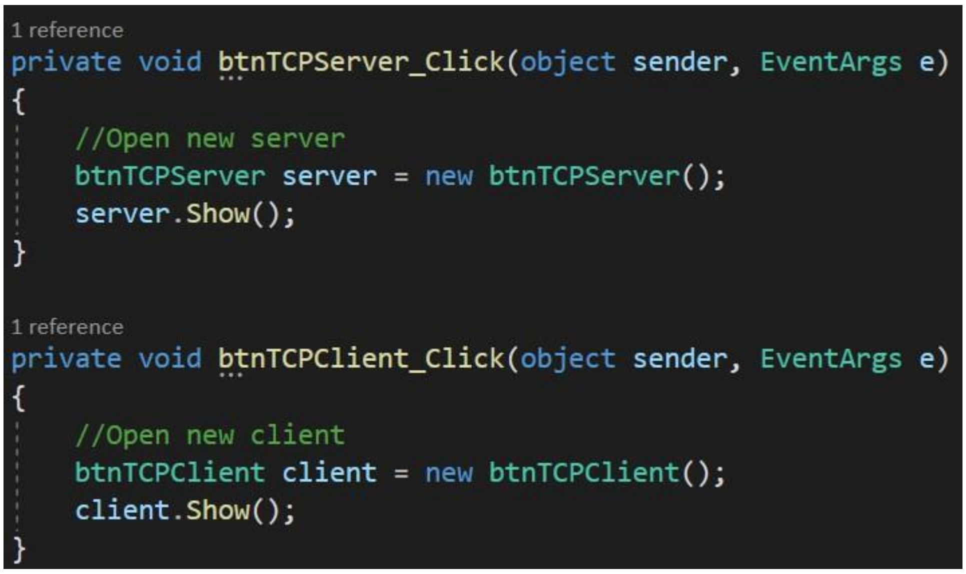
- Step 1: Design the interface which includes two buttons: one to open the TCP Server and one to open the TCP Client.
  The TCP Server interface includes a 'Listen' button for the server to start listening for client connections.
  The TCP Client interface includes three buttons: 'Connect' (to establish a connection with the server), 'Send Message' (to send a default message "Hello server"), and 'Close' (to close the connection to the server).


Client
Open TCP Server
Open TCP Client


TCP Server
Listen

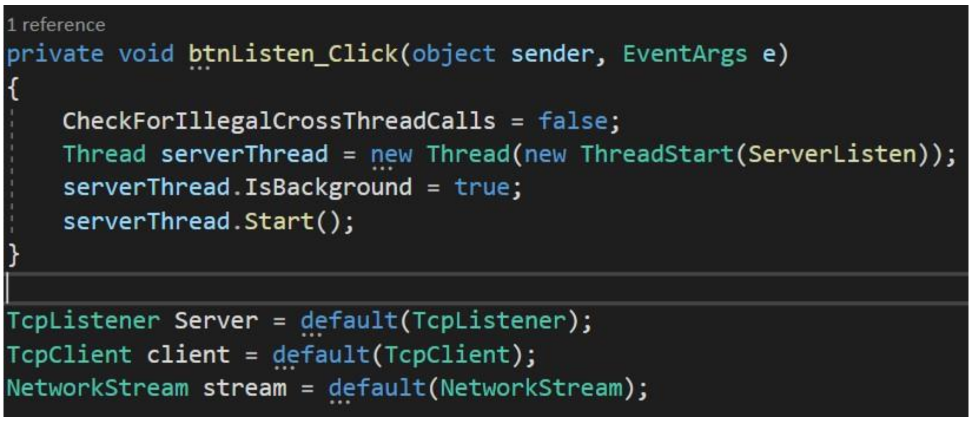


- Step 2: Code the functionality for the buttons on the forms
- For the initial form, use the show function to display new forms after the user clicks a button.



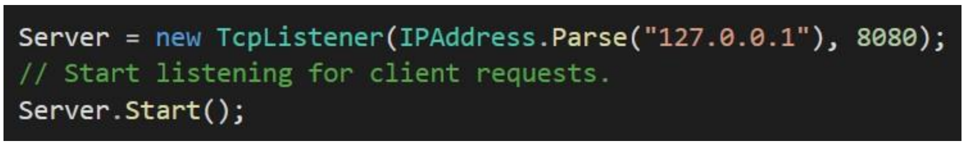
```
1 reference
private void btnTCPServer_Click(object sender, EventArgs e)
{
    //Open new server
    btnTCPServer server = new btnTCPServer();
    server.Show();
}

1 reference
private void btnTCPClient_Click(object sender, EventArgs e)
{
    //Open new client
    btnTCPClient client = new btnTCPClient();
    client.Show();
}
```

Form TCP Server: code the functionality of the 'Listen' button by first creating a thread (which stops when the form is closed) and establishing global variables.



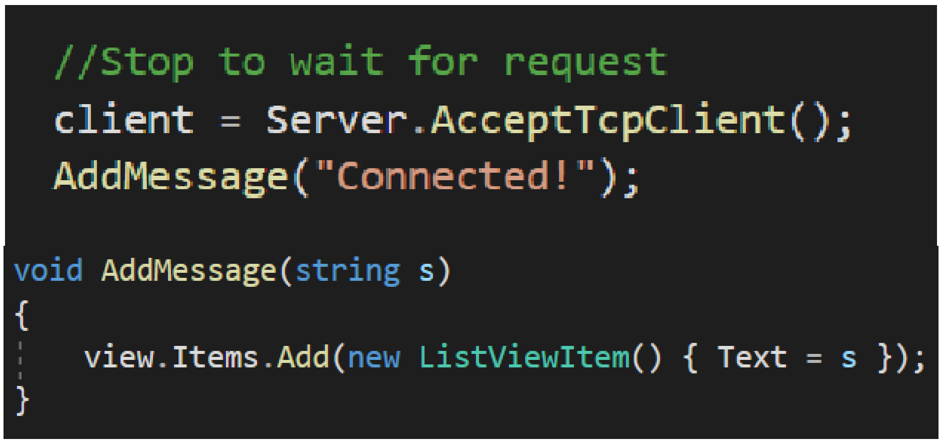
```
1 reference
private void btnListen_Click(object sender, EventArgs e)
{
    CheckForIllegalCrossThreadCalls = false;
    Thread serverThread = new Thread(new ThreadStart(ServerListen));
    serverThread.IsBackground = true;
    serverThread.Start();
}

TcpListener Server = default(TcpListener);
TcpClient client = default(TcpClient);
NetworkStream stream = default(NetworkStream);
```

Then, write the ServerListen function where the server starts listening for connections on the loopback IP address 127.0.0.1 at port 8080.



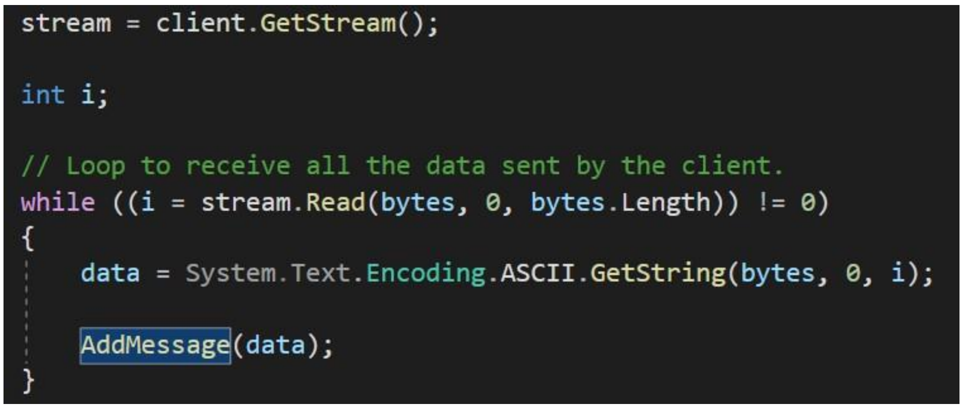
```
Server = new TcpListener(IPAddress.Parse("127.0.0.1"), 8080);
// Start listening for client requests.
Server.Start();
```

The server will then wait for a connection request from the client and, if successful, the screen will display "Connected" using the AddMessage function.



```
//Stop to wait for request
client = Server.AcceptTcpClient();
AddMessage("Connected!");

void AddMessage(string s)
{
    view.Items.Add(new ListViewItem() { Text = s });
}
```

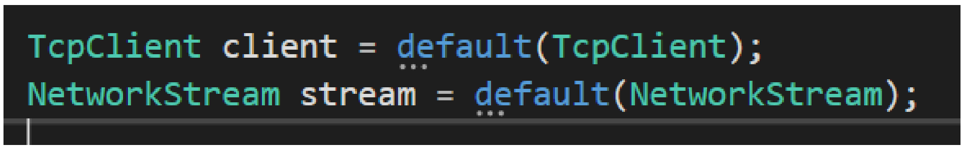
After the connection is established, create a loop to receive all information sent by the client and display it on the screen.



```
stream = client.GetStream();

int i;

// Loop to receive all the data sent by the client.
while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
{
    data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);

    AddMessage(data);
}
```

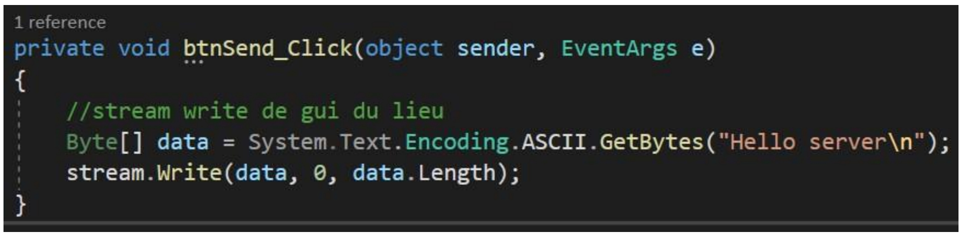
The server will close the connection when the client closes the connection or when the form is closed.

For the Form TCP Client: create two global variables.



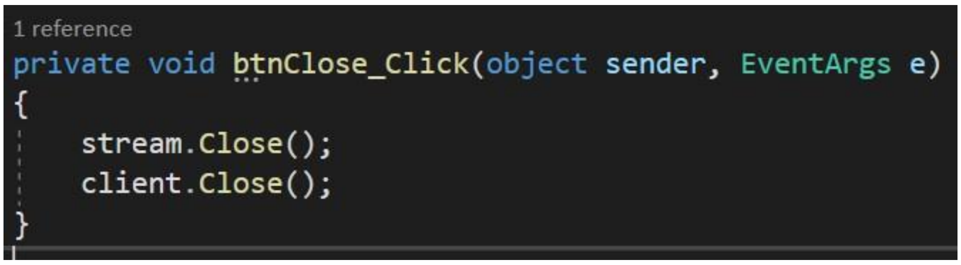
```
TcpClient client = default(TcpClient);
NetworkStream stream = default(NetworkStream);
```

- Code the functionality for the 'Send' button to send the default message "Hello server" through the stream



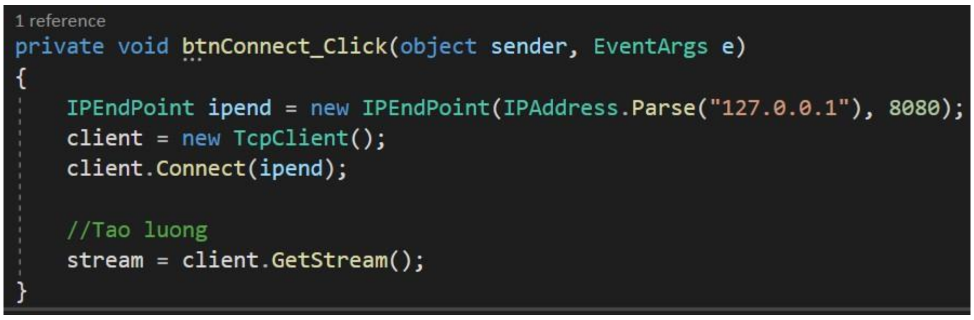
```
1 reference
private void btnSend_Click(object sender, EventArgs e)
{
    //stream write de gui du lieu
    Byte[] data = System.Text.Encoding.ASCII.GetBytes("Hello server\n");
    stream.Write(data, 0, data.Length);
}
```

- Code the functionality for the 'Close' button to close the stream and the connection to the server.



```
1 reference
private void btnClose_Click(object sender, EventArgs e)
{
    stream.Close();
    client.Close();
}
```

- Lastly, code the functionality for the 'Connect' button to establish a connection to the loopback IP address 127.0.0.1 at port 8080 using the connect method available in the TcpClient class.



```
1 reference
private void btnConnect_Click(object sender, EventArgs e)
{
    IPEndPoint ipend = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8080);
    client = new TcpClient();
    client.Connect(ipend);

    //Tao luong
    stream = client.GetStream();
}
```

4. Scenario 04

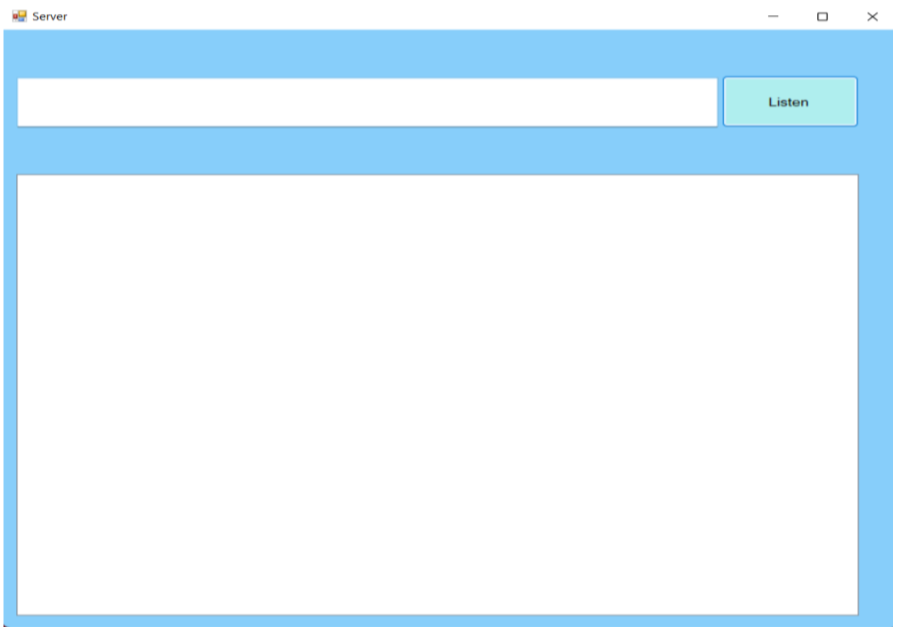
- **Resource:** Attached compressed file

- **Objective:**
  1 Server – Multi Client – Viết chương trình Chat Room / Gửi và nhận dữ liệu sử dụng TCP Client và TCP Listener. Mỗi người dùng sẽ có một tài khoản, khi một người dùng gửi tin nhắn thì tất cả mọi người còn lại đều sẽ nhận được tin nhắn đó.

- **Steps to implement:**

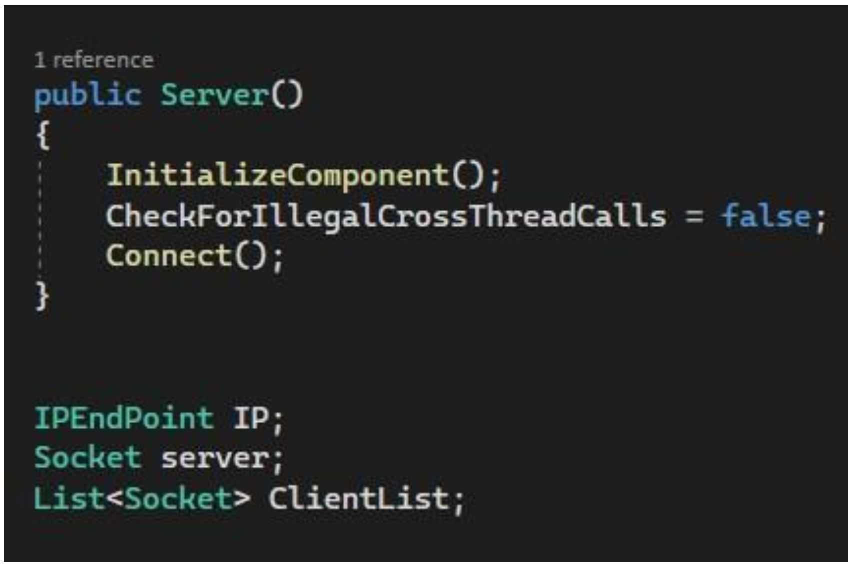
– Step 1: Design the interface

+ For Server: An optimized and simplified interface that includes only a Listen button to receive and listen to data from Clients, and a listView to display information about which server has connected to the server and what messages they have sent (a common chatroom).

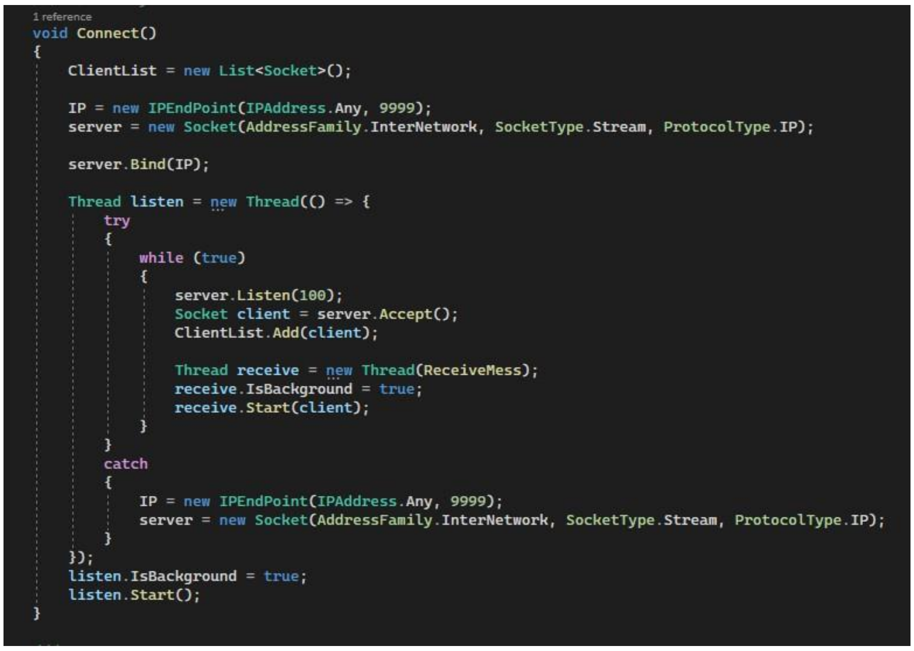


– Step 2: Proceed with coding

+ For the Send button:

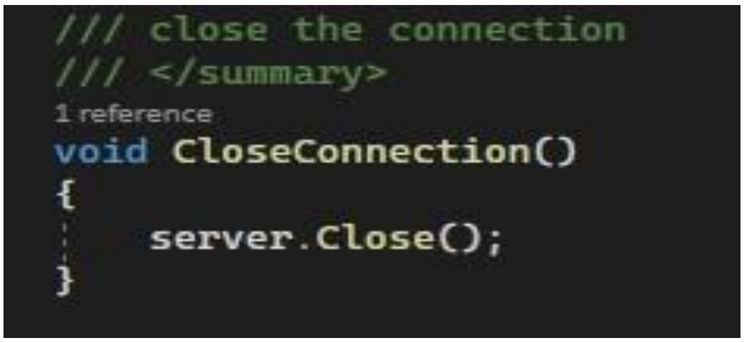


```
1 reference
public Server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    Connect();
}


IPEndPoint IP;
Socket server;
List<Socket> ClientList;
```



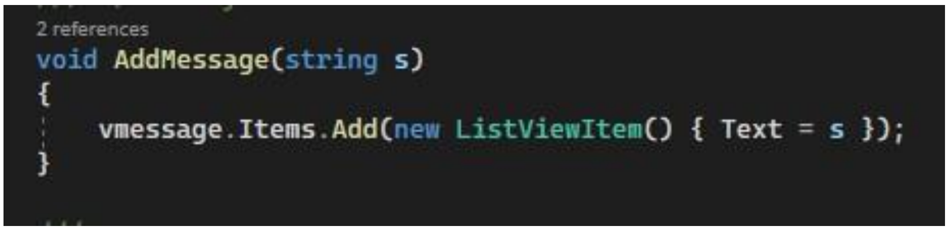
```
1 reference
void Connect()
{
    ClientList = new List<Socket>();

    IP = new IPEndPoint(IPAddress.Any, 9999);
    server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);

    server.Bind(IP);

    Thread listen = new Thread(() => {
        try
        {
            while (true)
            {
                server.Listen(100);
                Socket client = server.Accept();
                ClientList.Add(client);

                Thread receive = new Thread(ReceiveMess);
                receive.IsBackground = true;
                receive.Start(client);
            }
        }
        catch
        {
            IP = new IPEndPoint(IPAddress.Any, 9999);
            server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);
        }
    });
    listen.IsBackground = true;
    listen.Start();
}
```

- Declare IPEndPoint, Socket variables, and the Connect() function to connect to the server.
- The Connect function is used to connect to IPEndPoint and Socket.
- Use try-catch to check if clients can connect to the server, whether the server has started earlier, if not, display a MessageBox notifying the error, unable to connect to the server.
- Create a new Thread named listen to establish a connection with the ReceiveMess function, which is the function that receives messages from the created servers.



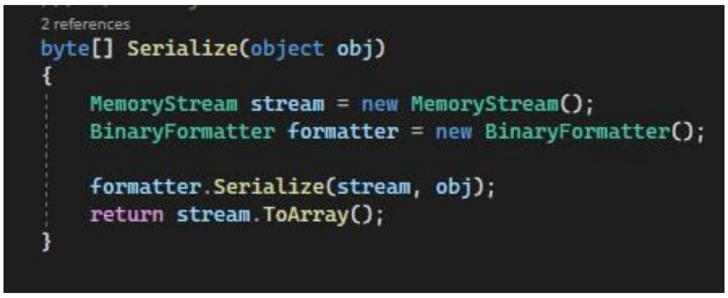
```
/// close the connection
/// </summary>
1 reference
void CloseConnection()
{
    server.Close();
}
```

- The CloseConnection function is used to close the client's connection.

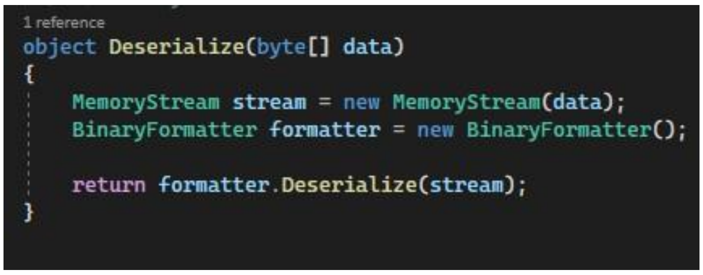


```
2 references
void AddMessage(string s)
{
    vmessage.Items.Add(new ListViewItem() { Text = s });
}
```

- The AddMessage function is used to add messages. These functions are similar to those on the server.

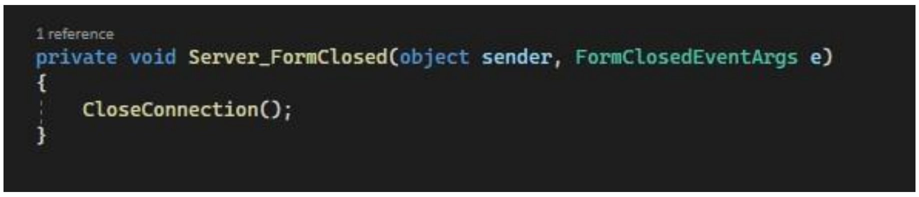


```
2 references
byte[] Serialize(object obj)
{
    MemoryStream stream = new MemoryStream();
    BinaryFormatter formatter = new BinaryFormatter();

    formatter.Serialize(stream, obj);
    return stream.ToArray();
}
```



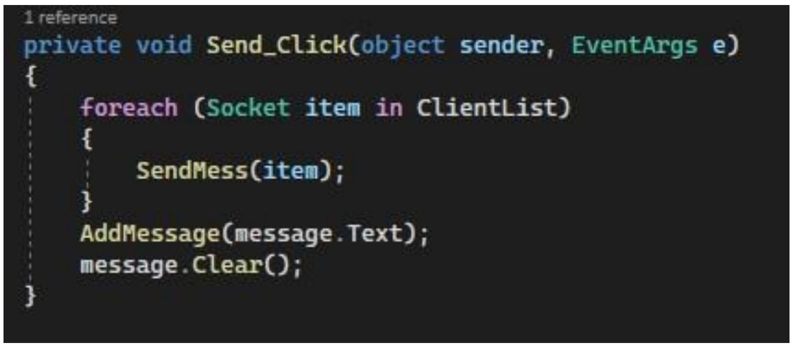
```
1 reference
object Deserialize(byte[] data)
{
    MemoryStream stream = new MemoryStream(data);
    BinaryFormatter formatter = new BinaryFormatter();

    return formatter.Deserialize(stream);
}
```

- The Serialize function is used for fragmentation and Deserialize for compiling fragments.



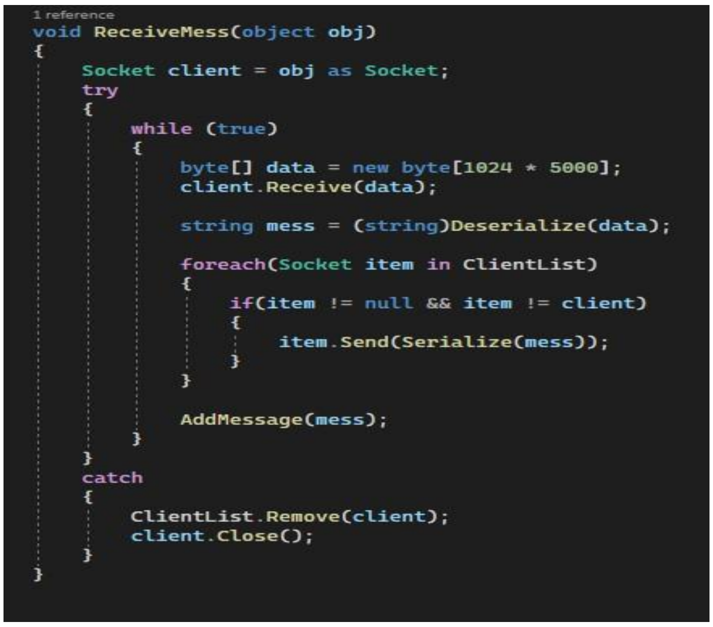
```
1 reference
private void Server_FormClosed(object sender, FormClosedEventArgs e)
{
    CloseConnection();
}
```

- Bắt sự kiên FormClosed để đóng Form Server.



```
1 reference
private void Send_Click(object sender, EventArgs e)
{
    foreach (Socket item in ClientList)
    {
        SendMess(item);
    }
    AddMessage(message.Text);
    message.Clear();
}
```

- The Button Listen is used to listen for messages from clients. When a message is received, the server will display it in the textbox message.Text.
- The .Clear() function is used to clear all text from the textbox message.Text.

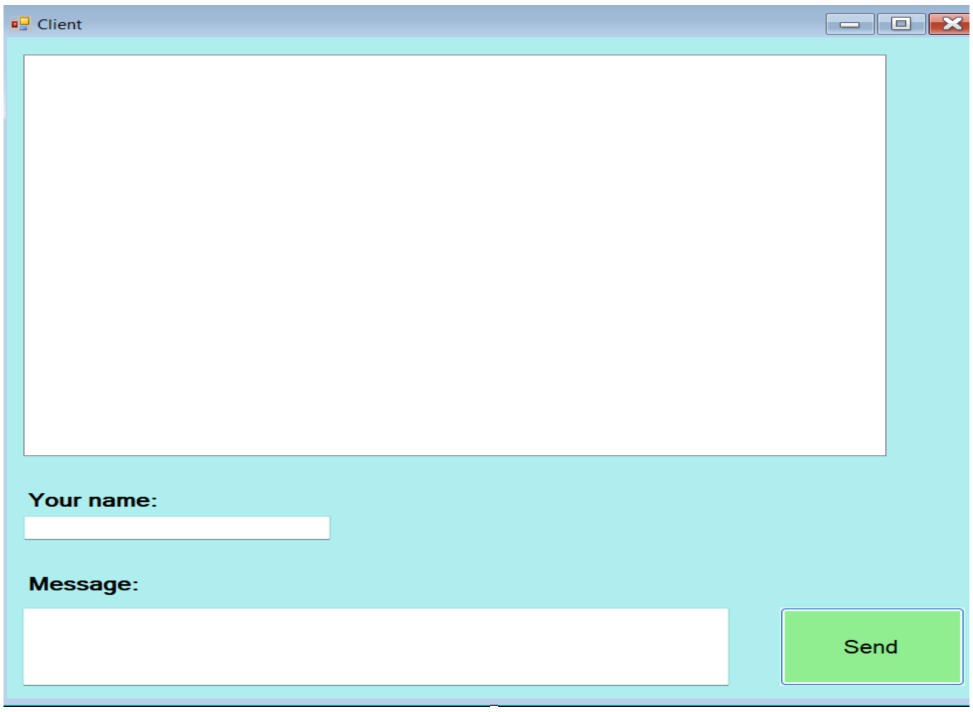


```
1 reference
void ReceiveMess(object obj)
{
    Socket client = obj as Socket;
    try
    {
        while (true)
        {
            byte[] data = new byte[1024 * 5000];
            client.Receive(data);

            string mess = (string)Deserialize(data);

            foreach(Socket item in ClientList)
            {
                if(item != null && item != client)
                {
                    item.Send(Serialize(mess));
                }
            }

            AddMessage(mess);
        }
    }
    catch
    {
        ClientList.Remove(client);
        client.Close();
    }
}
```

- The ReceiveMess function is used to receive messages from clients. The function takes a byte array as input and returns a string. The string contains the message that was received from the client.
- The data byte[] array is used to store the received message. The array is initialized with a size that is large enough to store the message.
- Khai báo chuỗi string mess để lưu các thông điệp từ phía các server gửi về. Hàm Deserialize(data) dùng để tổng hợp các thông điệp lại với nhau để tiện cho việc hiển thị trên màn hình (tổng hợp mảnh).
- The AddMessage function is used to add a message to the string mess. The function takes a string as input and adds it to the end of the string mess.

+ Client:

The txt_Name and txt_Message textboxes are used to enter the client's name and the message that the client wants to send. The Send button is used to send the message to the server. The button also includes the client's name so that the server can display who sent the message. The listView is used to display the clients and their messages.



END