

LAB MANUAL

Network Programming (108157)

**Muhammad Umar Khan (10619)**

**Contents**

[Lab 1 3](#_bookmark0)

OOP pillars implementation with output………………………………………………………………………………………

[Lab2 8](#_bookmark5)

[Create a first client and server application using socket programming (console) 8](#_bookmark6)

[Lab3a 10](#_bookmark7)

[Create a first client and server application using socket programming (Winforms) 10](#_bookmark8)

3.1-3.4……………………………………………………………………………………………………………………………………….11

[Lab3b 11](#_bookmark9)

[Create a chat Application for client server communication 11](#_bookmark10)

[Lab4 14](#_bookmark11)

[Multithreading and Delegates 14](#_bookmark12)

[Lab5 Connection oriented Sockets 40](#_bookmark13)

[Lab6 Create server that can communicated with multiple clients (Multi-Threaded Sever) 44](#_bookmark14)

[Lab 7: Connection less Sockets and Helper Classes 46](#_bookmark15)

[Ch #6 (6.1-6.4) 46](#_bookmark16)

[Lab 8: Asynchronous programming 49](#_bookmark17)

Lab 9: SMTP……………………………………………………………………………………………………………………………….50

Lab 10: FTP…………………………………………………………………………………………………………………………………51

Lab 11: Voice Call……………………………………………………………………………………………………………………….52

LAB#01 (OOP PILLARS)

**CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Threading;

namespace Hotelmanagementsystem

{

    static class Program

    {

        /// <summary>

        /// The main entry point for the application.

        /// </summary>

        [STAThread]

        static void Main()  //----------------------------Entry-Point-----------------------------------------//

        {

            SingleRoom SR = new SingleRoom();

            SR.Price();

            DoubleRoom DR = new DoubleRoom();

            DR.Price();

            FamilyRoom FR = new FamilyRoom();

            FR.Price();

            DecideMealPlan DMP = new DecideMealPlan("AmericanPlan");

            DecideMealPlan DMP1 = new DecideMealPlan("BednBreakfast", "EuropeanPlan", "\_BednBreakfast");

            Economy E = new Economy();

            E.Bill();

            Business B = new Business();

            B.Bill();

            VIP V = new VIP();

            V.Bill();

        }

    }

    //--------------------------------------------Encapsulation---------------------------------------------------------------------//

    class DecideMealPlan

    {

        private string \_AmericanPlan;

        private string \_BednBreakfast;

        private string \_ContinentalPlan;

        private string \_EuropeanPlan;

        public DecideMealPlan(string AmericanPlan)

        {

            this.\_AmericanPlan = AmericanPlan;

            if(AmericanPlan==\_AmericanPlan)

            {

                Console.WriteLine("The cost charged is 5000");

            }

        }

        public DecideMealPlan(string BednBreakfast,string ContinentalPlan,string EuropeanPlan)

        {

            this.\_BednBreakfast = BednBreakfast;

            this.\_ContinentalPlan = ContinentalPlan;

            this.\_EuropeanPlan = EuropeanPlan;

            if(\_BednBreakfast==BednBreakfast)

            {

                Console.WriteLine("The cost charged is 5000");

            }

            if(ContinentalPlan==\_ContinentalPlan)

            {

                Console.WriteLine("The cost charged is 8000");

            }

            if(EuropeanPlan==\_EuropeanPlan)

            {

                Console.WriteLine("The cost charged is 10000");

            }

        }

    }

    //--------------------------------------------Abstraction & Inheritance---------------------------------------------------------------------//

    public abstract class Room

    {

        public abstract double Price();

        protected double SingleRoomCharges = 8000;

    }

    public class SingleRoom : Room

    {

        public override double Price()

        {

            return SingleRoomCharges \* 1;

        }

    }

        public class DoubleRoom : Room

        {

        public override double Price()

            {

            return SingleRoomCharges \* 2;

            }

        }

        public class FamilyRoom : Room

        {

        public override double Price()

        {

            return SingleRoomCharges \* 3;

        }

    }

    //--------------------------------------------Polymorphism---------------------------------------------------------------------//

    class CustomerType

    {

        protected double BillCharge = 20000;

        public virtual double Bill()

        {

            return BillCharge \* 0;

        }

    }

    class Economy:CustomerType

    {

        public override double Bill()

        {

            return BillCharge \* 1;

        }

    }

    class Business:CustomerType

    {

        public override double Bill()

        {

            return BillCharge \* 2;

        }

    }

    class VIP:CustomerType

    {

        public override double Bill()

        {

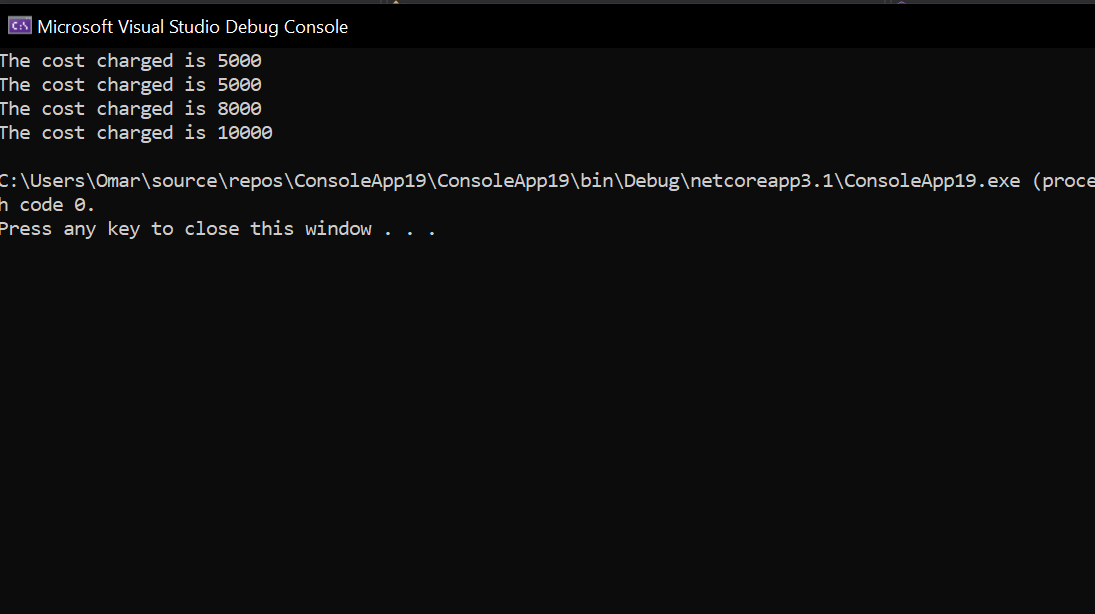
            return BillCharge \* 3;

        }

    }

    }

**OUTPUT:**

****

LAB#02 (CLIENT SERVER CONNECTION)

**CLIENT SIDE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApplication112

/// <summary>

/// ///////////client side code

/// </summary>

{

    class Program

    {

        static void Main(string[] args)

        {

            IPAddress ip = IPAddress.Parse("192.168.0.107");

            IPEndPoint ep = new IPEndPoint(ip, 2000);

            Socket sp = new Socket(ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

            sp.Connect(ep);

            Console.WriteLine("CONNECTED WITH THE SERVER");

            Console.ReadKey();

        }

    }

}

**SERVER SIDE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApplication112

{

    /// <summary>

    /// ///////////server side code

    /// </summary>

    class Program

    {

        static void Main(string[] args)

        {

            IPAddress ip = IPAddress.Parse("192.168.0.107");

            IPEndPoint ep = new IPEndPoint(ip, 2000);

            Socket sp = new Socket(ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

            sp.Bind(ep);

            sp.Listen(1);

            Console.WriteLine("waiting for the client");

            Socket c1 = sp.Accept();

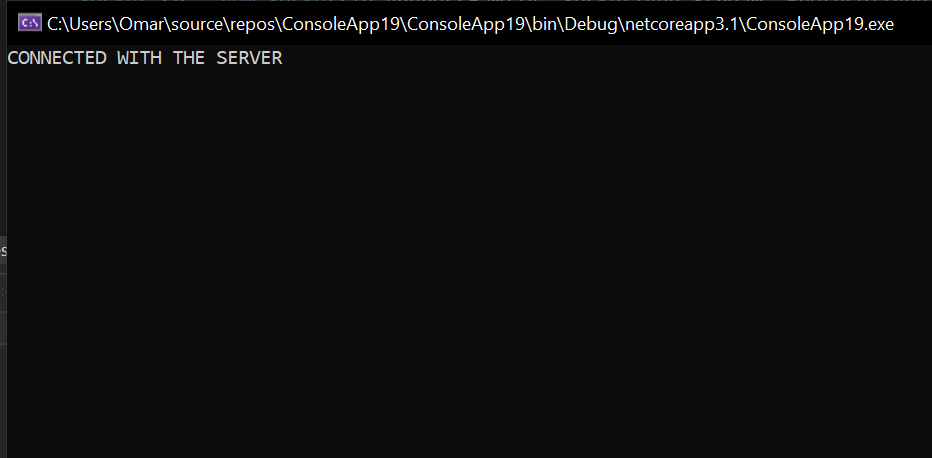
            Console.WriteLine("connected with the server");

        }

    }

}

**OUTPUT:**

****

**SENDING MESSAGES BY LOOP:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace Client

{

    class Program

    {

        static void Main(string[] args)

        {

            /// <summary>

            /// //////client side

            /// </summary>

            IPAddress ip = IPAddress.Parse("192.168.1.121");

            IPEndPoint eq = new IPEndPoint(ip, 2000);

            Socket sk = new Socket(ip.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

            sk.Connect(eq);

            Console.WriteLine("Client Connected");

            string str1 = "Mustafa";

            string str2 = "Omer";

            string str3 = "Shagil";

            string str4 = "Faisal";

            string str5 = "Shoaib";

            string[] arr = { str1, str2, str3, str4, str5 };

            int i = 0;

            while (i < 5)

            {

                sk.Send(Encoding.ASCII.GetBytes(arr[i]));

                i++;

            }

            Console.ReadKey();

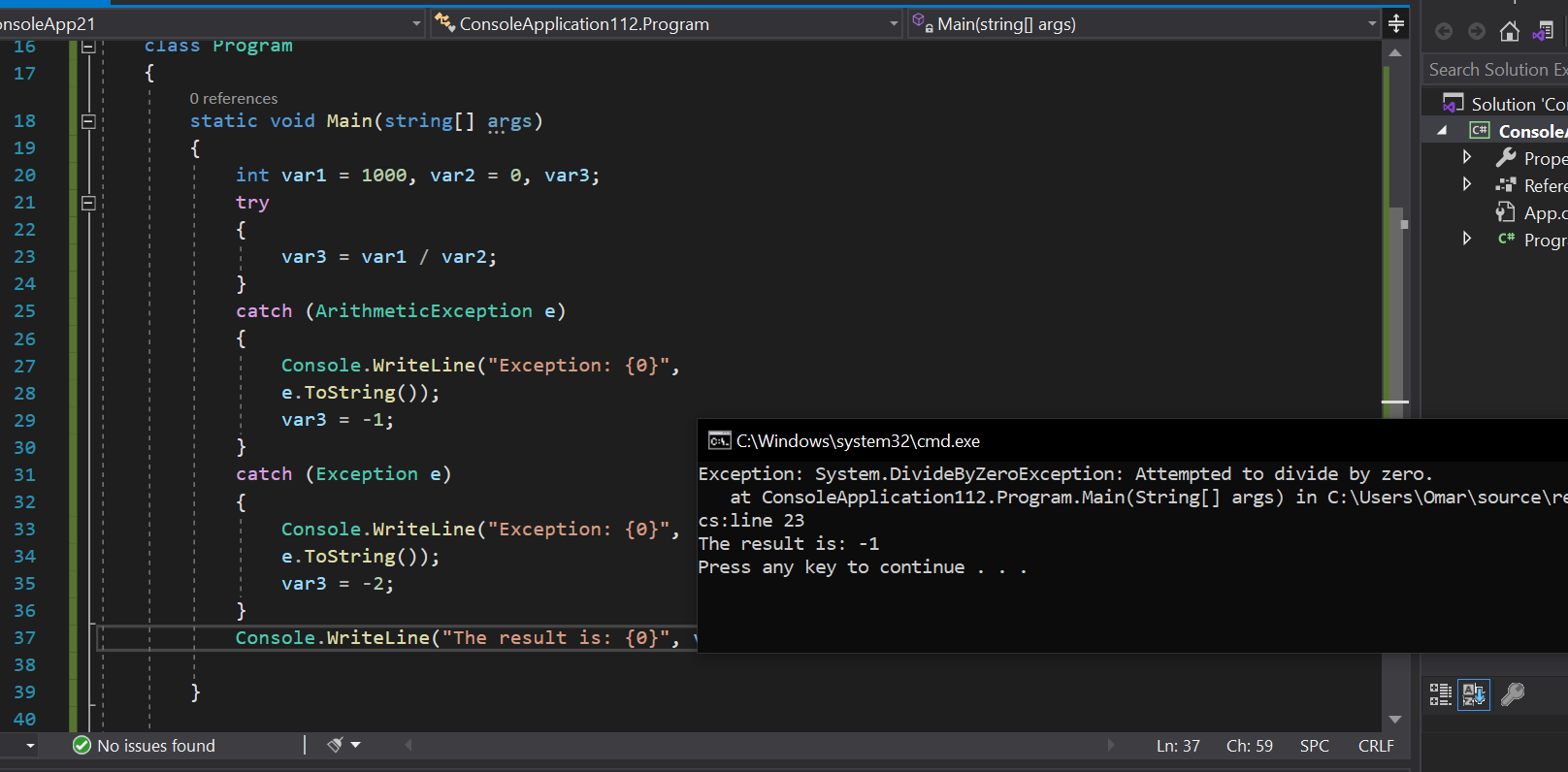
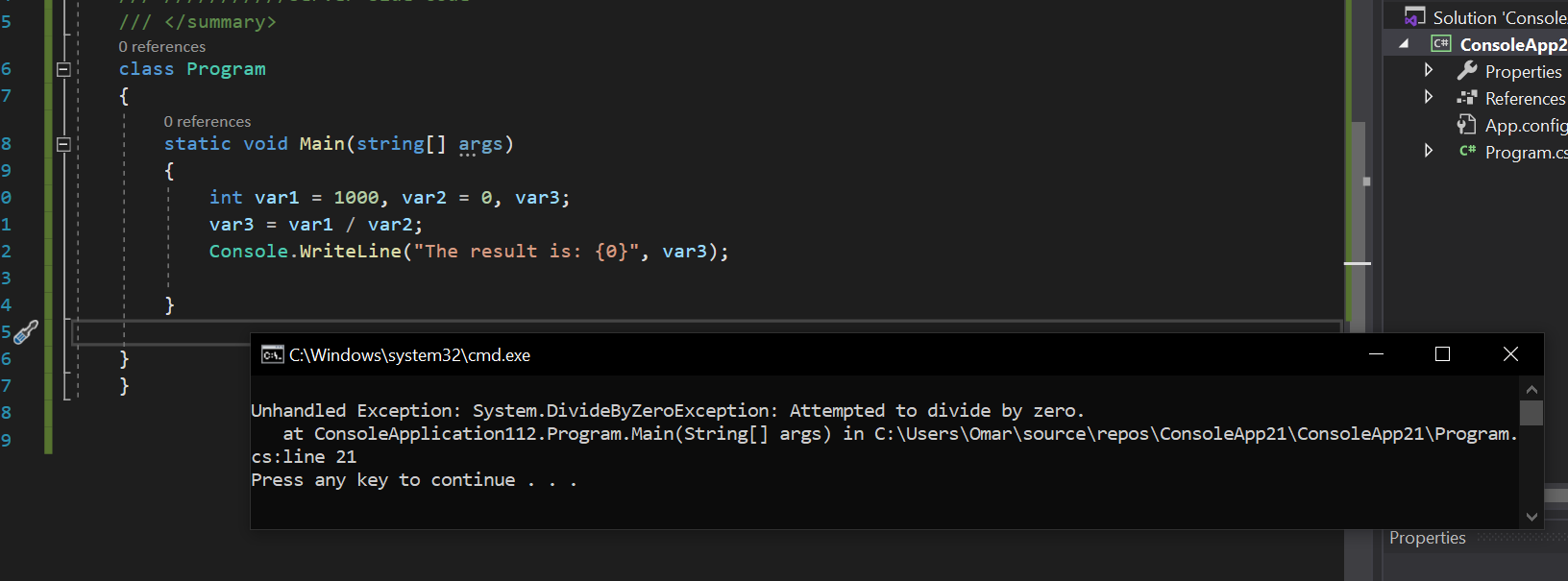
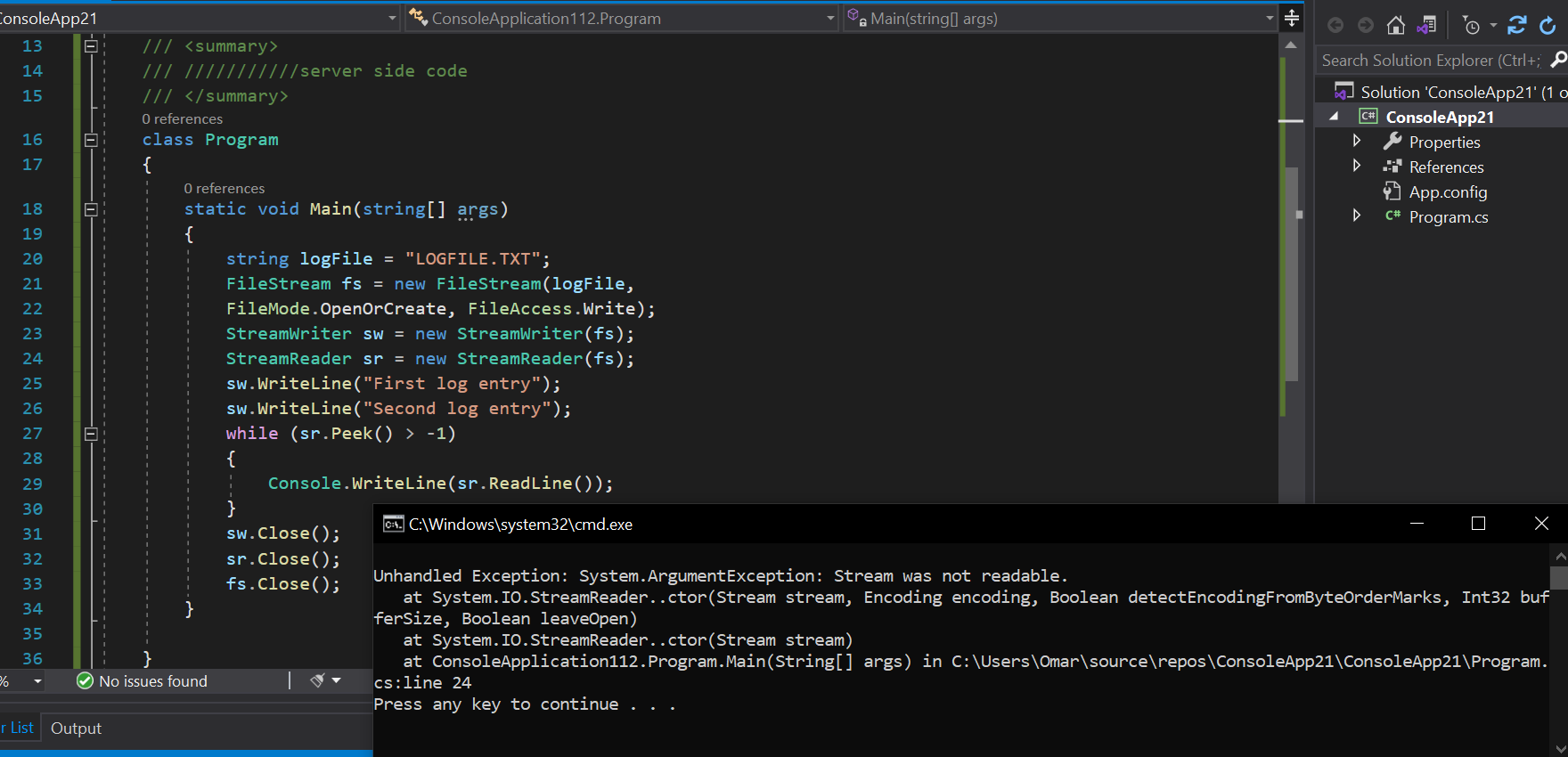
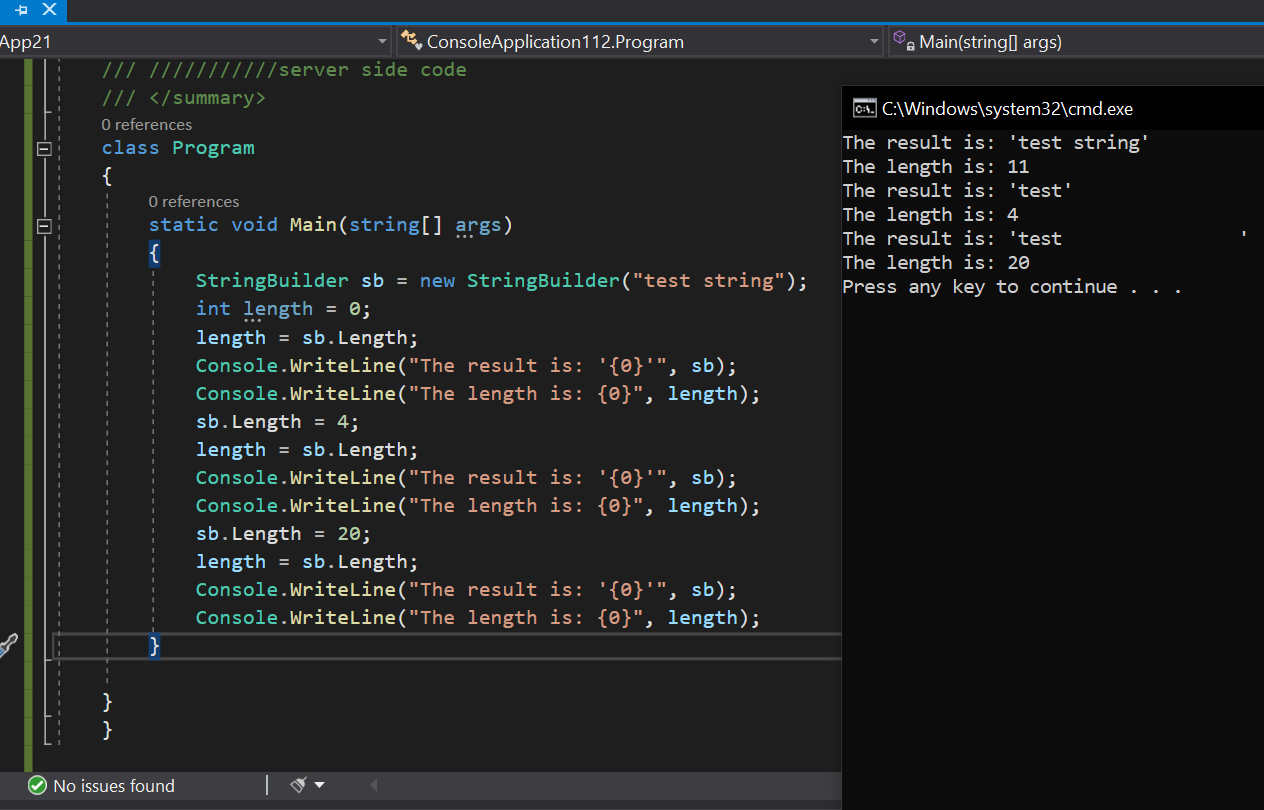
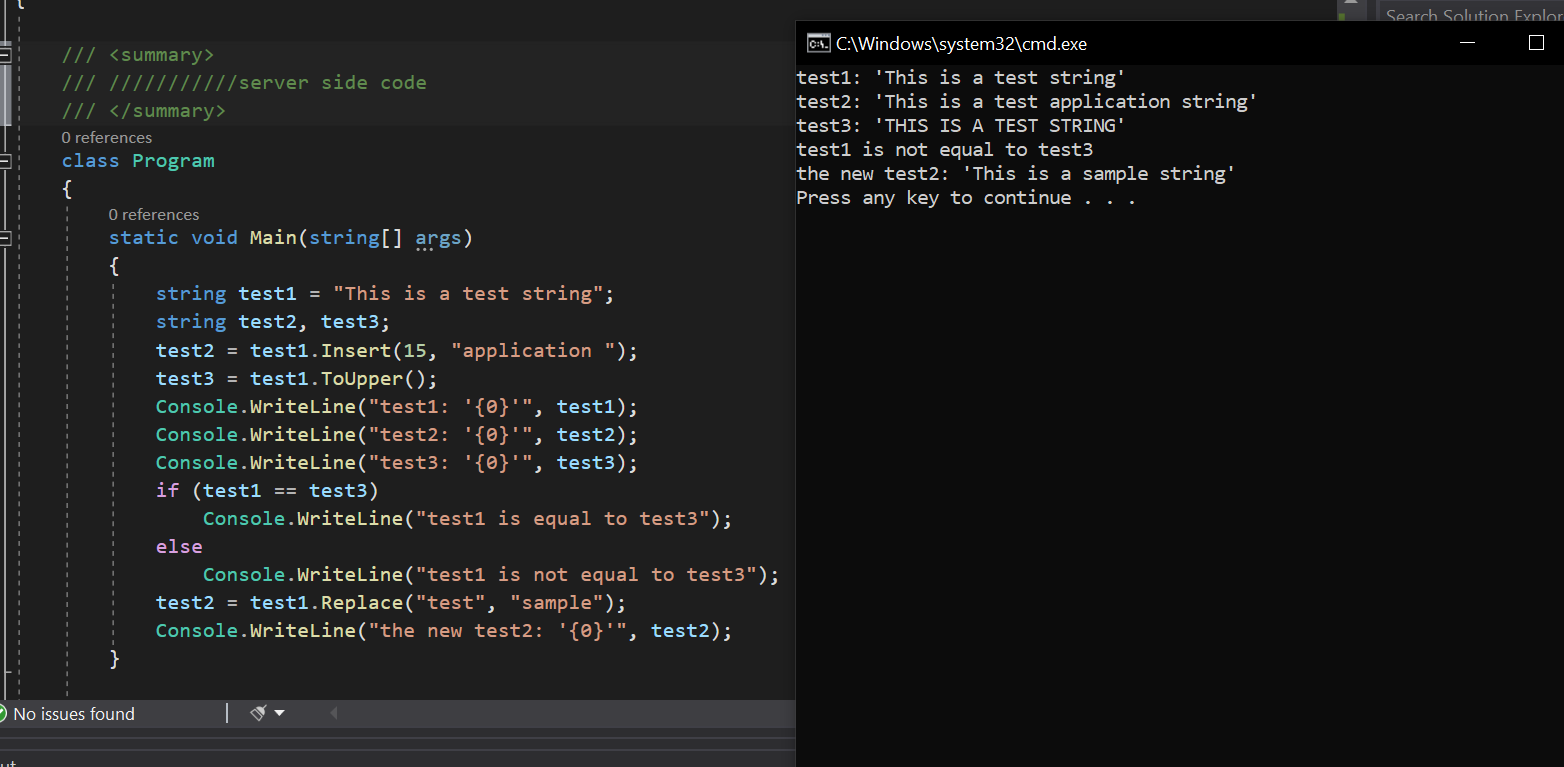
        }

    }

}

LAB#03 (CHAPTER ONE LISTINGS)

**(1.4-1.8)**



LAB#04(MULTITHREADS LISTING)

**SERVER SIDE CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using System.Threading;

namespace Server

{

    class Program                                          //SERVER SIDE

    {

        static void Main(string[] args)

        {

            Thread t = new Thread(delegate ()

            {

                Server myserver = new Server("192.168.0.107", 2100);

            });

            t.Start();

        }

    }

    class Server

    {

        TcpListener server = null;

        public Server(string ip, int port)

        {

            IPAddress localAddr = IPAddress.Parse(ip);

            server = new TcpListener(localAddr, port);

            server.Start();

            StartListener();

        }

        public void StartListener()

        {

            try

            {

                while (true)

                {

                    Console.WriteLine("Waiting for a connection...");

                    TcpClient client = server.AcceptTcpClient();

                    Console.WriteLine("Connected!");

                    Thread t = new Thread(new ParameterizedThreadStart(HandleDevice));

                    t.Start(client);

                }

            }

            catch (SocketException e)

            {

                Console.WriteLine("Socket Exception : {0}", e);

            }

        }

        public void HandleDevice(Object obj)

        {

            TcpClient client = (TcpClient)obj;

            var stream = client.GetStream();

            string imei = string.Empty;

            string data = null;

            Byte[] bytes = new Byte[1024];

            int i;

            try

            {

                while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)

                {

                    string hex = BitConverter.ToString(bytes);

                    data = Encoding.ASCII.GetString(bytes, 0, i);

                    Console.WriteLine("{1}: Received: {0}", data, Thread.CurrentThread.ManagedThreadId);

                    string str = "hello";

                    Byte[] reply = Encoding.ASCII.GetBytes(data);

                    stream.Write(reply, 0, reply.Length);

                    Console.WriteLine("{1}: Sent: {0}", data, Thread.CurrentThread.ManagedThreadId);

                }

            }

            catch (Exception e)

            {

                Console.WriteLine("Exception: {0}", e.ToString());

                client.Close();

            }

        }

    }

}

server side

--------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using System.Threading;

namespace Server

{

    class Program                                          //SERVER SIDE

    {

        static void Main(string[] args)

        {

            Thread t = new Thread(delegate ()

            {

                Server myserver = new Server("192.168.0.107", 2100);

            });

            t.Start();

        }

    }

    class Server

    {

        TcpListener server = null;

        public Server(string ip, int port)

        {

            IPAddress localAddr = IPAddress.Parse(ip);

            server = new TcpListener(localAddr, port);

            server.Start();

            StartListener();

        }

        public void StartListener()

        {

            try

            {

                while (true)

                {

                    Console.WriteLine("Waiting for a connection...");

                    TcpClient client = server.AcceptTcpClient();

                    Console.WriteLine("Connected!");

                    Thread t = new Thread(new ParameterizedThreadStart(HandleDevice));

                    t.Start(client);

                }

            }

            catch (SocketException e)

            {

                Console.WriteLine("Socket Exception : {0}", e);

            }

        }

        public void HandleDevice(Object obj)

        {

            TcpClient client = (TcpClient)obj;

            var stream = client.GetStream();

            string imei = string.Empty;

            string data = null;

            Byte[] bytes = new Byte[1024];

            int i;

            try

            {

                while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)

                {

                    string hex = BitConverter.ToString(bytes);

                    data = Encoding.ASCII.GetString(bytes, 0, i);

                    Console.WriteLine("{1}: Received: {0}", data, Thread.CurrentThread.ManagedThreadId);

                    string str="hello";

                    Byte[] reply = Encoding.ASCII.GetBytes(data);

                    stream.Write(reply, 0, reply.Length);

                    Console.WriteLine("{1}: Sent: {0}", data, Thread.CurrentThread.ManagedThreadId);

                }

            }

            catch (Exception e)

            {

                Console.WriteLine("Exception: {0}", e.ToString());

                client.Close();

            }

        }

    }

}

----------------------------------------------------------------------------------------------

client side

--------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Sockets;

using System.Threading;

namespace ConsoleApplication45

{

    class Program                   //CLIENT SIDE

    {

        static void Main(string[] args)

        {

            string a;

            string b;

            Console.WriteLine("Enter the first string you would like to send then recieve it afterwards:\n");

            a = Console.ReadLine();

            Console.WriteLine("Enter the second string you would like to send then recieve it afterwards:\n");

            b = Console.ReadLine();

            Console.WriteLine("Code is creating 2 clients in separate threads & both clients will send 3 messages with the Sleep of 2 seconds after each message.");

           new Thread(() =>

            {

                Thread.CurrentThread.IsBackground = true;

                Connect("192.168.0.107", a );

            }).Start();

            new Thread(() =>

            {

                Thread.CurrentThread.IsBackground = true;

                Connect("192.168.0.107", b);

            }).Start();

            Console.ReadLine();

        }

        static void Connect(String server, String message)

        {

            try

            {

                Int32 port = 2100;

                TcpClient client = new TcpClient(server, port);

                NetworkStream stream = client.GetStream();

                int count = 0;

                while (count++ < 3)

                {

                    // Translate the Message into ASCII.

                    Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

                    // Send the message to the connected TcpServer.

                    stream.Write(data, 0, data.Length);

                    Console.WriteLine("Sent: {0}", message);

                    // Bytes Array to receive Server Response.

                    data = new Byte[256];

                    String response = String.Empty;

                    // Read the Tcp Server Response Bytes.

                    Int32 bytes = stream.Read(data, 0, data.Length);

                    response = System.Text.Encoding.ASCII.GetString(data, 0, bytes);

                    Console.WriteLine("Received: {0}", response);

                    Thread.Sleep(2000);

                }

                stream.Close();

                client.Close();

            }

            catch (Exception e)

            {

                Console.WriteLine("Exception: {0}", e);

            }

            Console.ReadLine();

        }

    }

}

**CLIENT SIDE CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net.Sockets;

using System.Threading;

namespace ConsoleApplication45

{

    class Program                   //CLIENT SIDE

    {

        static void Main(string[] args)

        {

            string a;

            string b;

            Console.WriteLine("Enter the first string you would like to send then recieve it afterwards:\n");

            a = Console.ReadLine();

            Console.WriteLine("Enter the second string you would like to send then recieve it afterwards:\n");

            b = Console.ReadLine();

            Console.WriteLine("Code is creating 2 clients in separate threads & both clients will send 3 messages with the Sleep of 2 seconds after each message.");

            new Thread(() =>

            {

                Thread.CurrentThread.IsBackground = true;

                Connect("192.168.0.107", a);

            }).Start();

            new Thread(() =>

            {

                Thread.CurrentThread.IsBackground = true;

                Connect("192.168.0.107", b);

            }).Start();

            Console.ReadLine();

        }

        static void Connect(String server, String message)

        {

            try

            {

                Int32 port = 2100;

                TcpClient client = new TcpClient(server, port);

                NetworkStream stream = client.GetStream();

                int count = 0;

                while (count++ < 3)

                {

                    // Translate the Message into ASCII.

                    Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

                    // Send the message to the connected TcpServer.

                    stream.Write(data, 0, data.Length);

                    Console.WriteLine("Sent: {0}", message);

                    // Bytes Array to receive Server Response.

                    data = new Byte[256];

                    String response = String.Empty;

                    // Read the Tcp Server Response Bytes.

                    Int32 bytes = stream.Read(data, 0, data.Length);

                    response = System.Text.Encoding.ASCII.GetString(data, 0, bytes);

                    Console.WriteLine("Received: {0}", response);

                    Thread.Sleep(2000);

                }

                stream.Close();

                client.Close();

            }

            catch (Exception e)

            {

                Console.WriteLine("Exception: {0}", e);

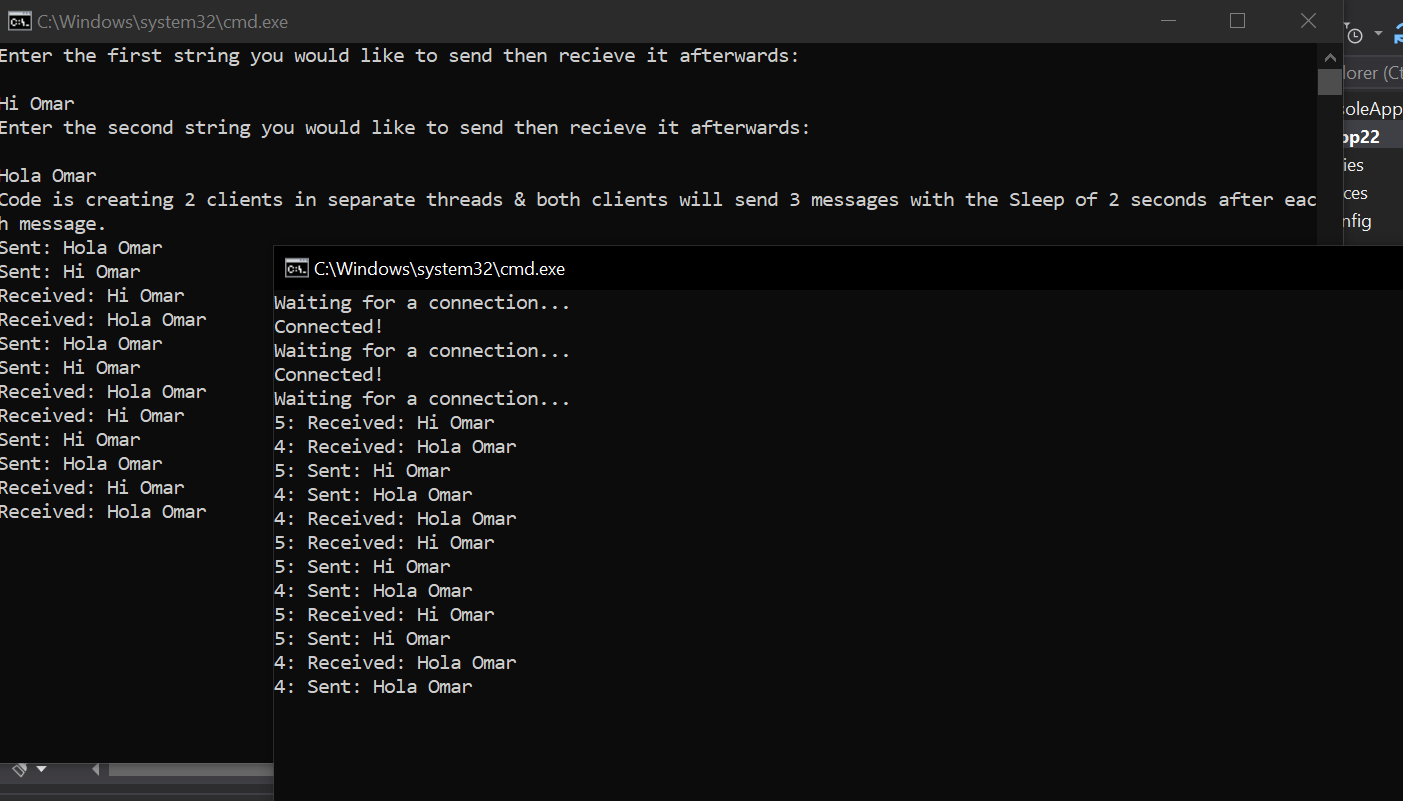
            }

            Console.ReadLine();

        }

    }

}

**OUTPUT:**

LAB#05(MULTITHREADS LISTING – CH#05 LISTINGS)

**CLIENT SIDE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApp24

{

    class Program

    {

        static void Main(string[] args)    //CLIENT SLIDE

        {

            byte[] data = new byte[1024];

            string input, stringData;

            IPEndPoint ipep = new IPEndPoint(

            IPAddress.Parse("192.168.0.107"), 9050);

            Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

            try

            {

                server.Connect(ipep);

            }

            catch (SocketException e)

            {

                Console.WriteLine("Unable to connect to server.");

                Console.WriteLine(e.ToString());

                return;

            }

            int recv = server.Receive(data);

            stringData = Encoding.ASCII.GetString(data, 0, recv);

            Console.WriteLine(stringData);

            while (true)

            {

                input = Console.ReadLine();

                if (input == "exit")

                    break;

                server.Send(Encoding.ASCII.GetBytes(input));

                data = new byte[1024];

                recv = server.Receive(data);

                stringData = Encoding.ASCII.GetString(data, 0, recv);

                Console.WriteLine(stringData);

            }

            Console.WriteLine("Disconnecting from server...");

            server.Shutdown(SocketShutdown.Both);

            server.Close();

        }

    }

}

**SERVER SIDE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApp23

{

    class Program

    {

        static void Main(string[] args)         //SERVER SIDE

        {

            int recv;

            byte[] data = new byte[1024];

            IPEndPoint ipep = new IPEndPoint(IPAddress.Any,

            9050);

            Socket newsock = new

            Socket(AddressFamily.InterNetwork,

            SocketType.Stream, ProtocolType.Tcp);

            newsock.Bind(ipep);

            newsock.Listen(10);

            Console.WriteLine("Waiting for a client...");

            Socket client = newsock.Accept();

            IPEndPoint clientep =

            (IPEndPoint)client.RemoteEndPoint;

            Console.WriteLine("Connected with {0} at port {1}",

            clientep.Address, clientep.Port);

            string welcome = "Welcome to my test server";

            data = Encoding.ASCII.GetBytes(welcome);

            client.Send(data, data.Length,

            SocketFlags.None);

            while (true)

            {

                data = new byte[1024];

                recv = client.Receive(data);

                if (recv == 0)

                    break;

                Console.WriteLine(

                Encoding.ASCII.GetString(data, 0, recv));

                client.Send(data, recv, SocketFlags.None);

            }

            Console.WriteLine("Disconnected from {0}",

            clientep.Address);

            client.Close();

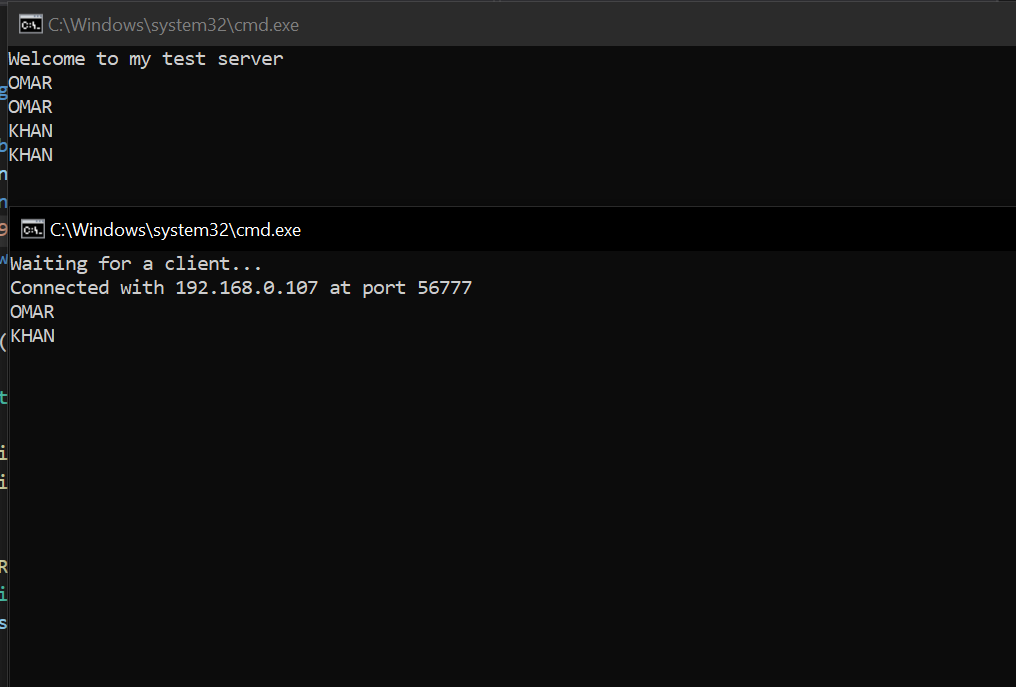
            newsock.Close();

        }

    }

}

**OUTPUT:**

****

**(5.3-5.4)**

**SERVER SIDE CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApp24

{

    class Program

    {

        static void Main(string[] args)    //SERVER SIDE

        {

            int recv;

            byte[] data = new byte[1024];

            IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);

            Socket newsock = new Socket(AddressFamily.InterNetwork,

            SocketType.Stream, ProtocolType.Tcp);

            newsock.Bind(ipep);

            newsock.Listen(10);

            Console.WriteLine("Waiting for a client...");

            Socket client = newsock.Accept();

            string welcome = "Welcome to my test server";

            data = Encoding.ASCII.GetBytes(welcome);

            client.Send(data, data.Length,

            SocketFlags.None);

            IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;

            Console.WriteLine("Connected with {0} at port {1}",

            newclient.Address, newclient.Port);

            for (int i = 0; i < 5; i++)

            {

                recv = client.Receive(data);

                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));

            }

            Console.WriteLine("Disconnecting from {0}", newclient.Address);

            client.Close();

            newsock.Close();

        }

    }

}

**CLIENT SIDE CODE:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApp25

{

    class Program

    {

        static void Main(string[] args)             //CLIENT SIDE

        {

            byte[] data = new byte[1024];

            string stringData;

            IPEndPoint ipep = new IPEndPoint(

            IPAddress.Parse("192.168.0.107"), 9050);

            Socket server = new Socket(AddressFamily.InterNetwork,

            SocketType.Stream, ProtocolType.Tcp);

            try

            {

                server.Connect(ipep);

            }

            catch (SocketException e)

            {

                Console.WriteLine("Unable to connect to server.");

                Console.WriteLine(e.ToString());

                return;

            }

            int recv = server.Receive(data);

            stringData = Encoding.ASCII.GetString(data, 0, recv);

            Console.WriteLine(stringData);

            server.Send(Encoding.ASCII.GetBytes("message 1"));

            server.Send(Encoding.ASCII.GetBytes("message 2"));

            server.Send(Encoding.ASCII.GetBytes("message 3"));

            server.Send(Encoding.ASCII.GetBytes("message 4"));

            server.Send(Encoding.ASCII.GetBytes("message 5"));

            Console.WriteLine("Disconnecting from server...");

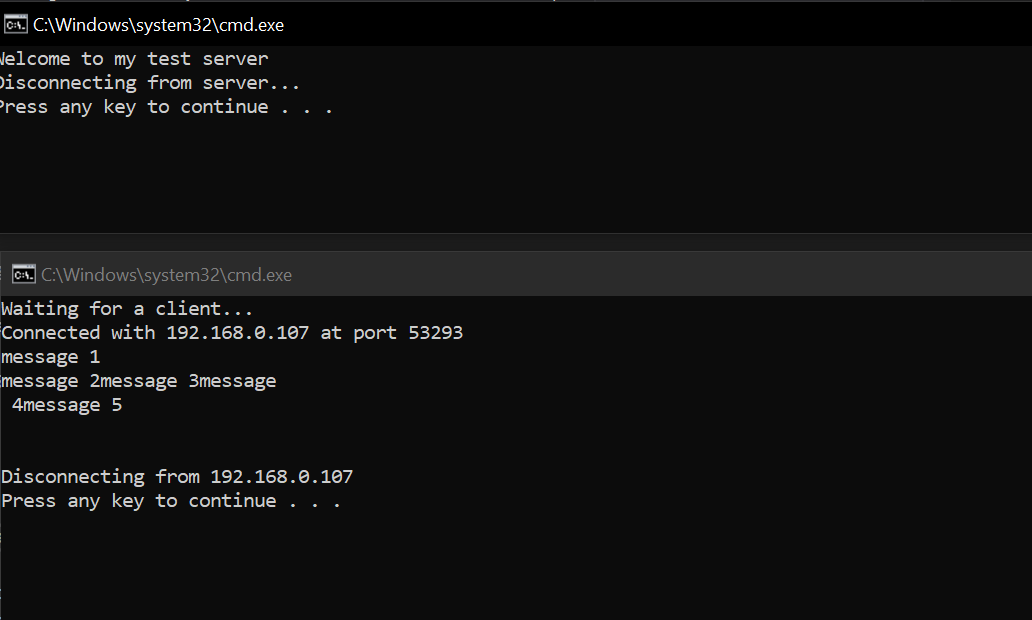
            server.Shutdown(SocketShutdown.Both);

            server.Close();

        }

    }

}

**OUTPUT:**

**(5.5-5.6)**

**SERVER SIDE CODE:**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class FixedTcpSrvr

{

    private static int SendData(Socket s, byte[] data)          //SERVER SIDE

    {

        int total = 0;

        int size = data.Length;

        int dataleft = size;

        int sent;

        while (total < size)

        {

            sent = s.Send(data, total, dataleft, SocketFlags.None);

            total += sent;

            dataleft -= sent;

        }

        return total;

    }

    private static byte[] ReceiveData(Socket s, int size)

    {

        int total = 0;

        int dataleft = size;

        byte[] data = new byte[size];

        int recv;

        while (total < size)

        {

            recv = s.Receive(data, total, dataleft, 0);

            if (recv == 0)

            {

                data = Encoding.ASCII.GetBytes("exit");

                break;

            }

            total += recv;

            dataleft -= recv;

        }

        return data;

    }

    public static void Main()

    {

        byte[] data = new byte[1024];

        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);

        Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        newsock.Bind(ipep);

        newsock.Listen(10);

        Console.WriteLine("Waiting for a client...");

        Socket client = newsock.Accept();

        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;

        Console.WriteLine("Connected with {0} at port {1}",

        newclient.Address, newclient.Port);

        string welcome = "Welcome to my test server";

        data = Encoding.ASCII.GetBytes(welcome);

        int sent = SendData(client, data);

        for (int i = 0; i < 5; i++)

        {

            data = ReceiveData(client, 9);

            Console.WriteLine(Encoding.ASCII.GetString(data));

        }

        Console.WriteLine("Disconnected from {0}", newclient.Address);

        client.Close();

        newsock.Close();

    }

}

**CLIENT SIDE CODE:**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class FixedTcpClient

{

    private static int SendData(Socket s, byte[] data)          //CLIENT SIDE

    {

        int total = 0;

        int size = data.Length;

        int dataleft = size;

        int sent;

        while (total < size)

        {

            sent = s.Send(data, total, dataleft, SocketFlags.None);

            total += sent;

            dataleft -= sent;

        }

        return total;

    }

    private static byte[] ReceiveData(Socket s, int size)

    {

        int total = 0;

        int dataleft = size;

        byte[] data = new byte[size];

        int recv;

        while (total < size)

        {

            recv = s.Receive(data, total, dataleft, 0);

            if (recv == 0)

            {

                data = Encoding.ASCII.GetBytes("exit ");

                break;

            }

            total += recv;

            dataleft -= recv;

        }

        return data;

    }

    public static void Main()

    {

        byte[] data = new byte[1024];

        int sent;

        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("192.168.0.107"), 9050);

        Socket server = new Socket(AddressFamily.InterNetwork,

        SocketType.Stream, ProtocolType.Tcp);

        try

        {

            server.Connect(ipep);

        }

        catch (SocketException e)

        {

            Console.WriteLine("Unable to connect to server.");

            Console.WriteLine(e.ToString());

            return;

        }

        int recv = server.Receive(data);

        string stringData = Encoding.ASCII.GetString(data, 0, recv);

        Console.WriteLine(stringData);

        sent = SendData(server, Encoding.ASCII.GetBytes("message 1"));

        sent = SendData(server, Encoding.ASCII.GetBytes("message 2"));

        sent = SendData(server, Encoding.ASCII.GetBytes("message 3"));

        sent = SendData(server, Encoding.ASCII.GetBytes("message 4"));

        sent = SendData(server, Encoding.ASCII.GetBytes("message 5"));

        Console.WriteLine("Disconnecting from server...");

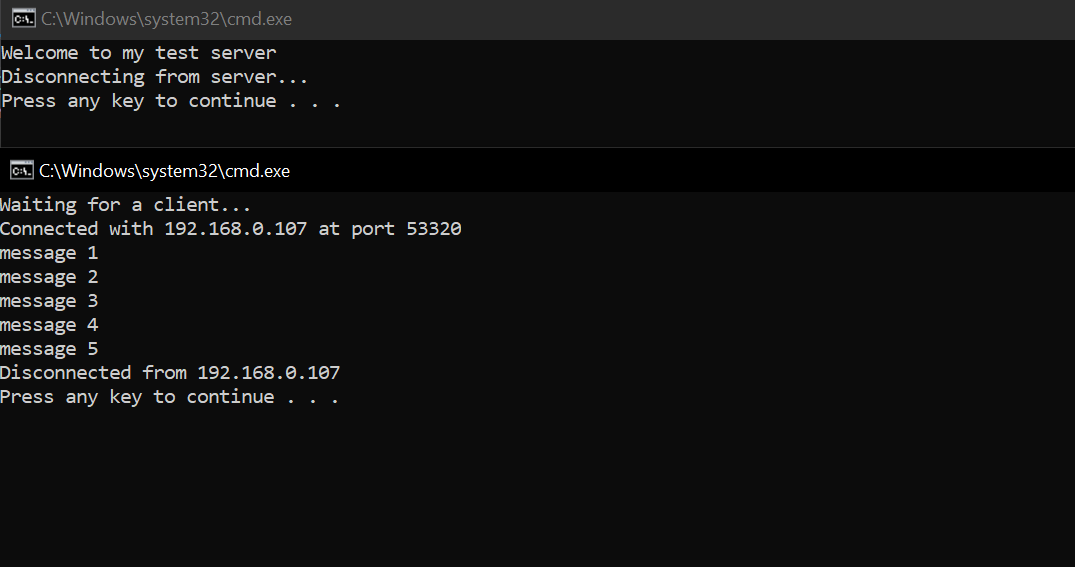
        server.Shutdown(SocketShutdown.Both);

        server.Close();

    }

}

**OUTPUT:**

****

(CH#7 HELPER CLASSES)

**(7.1-7.2)**

**CLIENT SIDE CODE:**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class TcpClientSample

{

    public static void Main()               //CLIENT SIDE

                                              //TCP CLIENT

    {

        byte[] data = new byte[1024];

        string input, stringData;

        TcpClient server;

        try

        {

            server = new TcpClient("192.168.0.107", 9050);

        }

        catch (SocketException)

        {

            Console.WriteLine("Unable to connect to server");

            return;

        }

        NetworkStream ns = server.GetStream();

        int recv = ns.Read(data, 0, data.Length);

        stringData = Encoding.ASCII.GetString(data, 0, recv);

        Console.WriteLine(stringData);

        while (true)

        {

            input = Console.ReadLine();

            if (input == "exit")

                break;

            ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);

            ns.Flush();

            data = new byte[1024];

            recv = ns.Read(data, 0, data.Length);

            stringData = Encoding.ASCII.GetString(data, 0, recv);

            Console.WriteLine(stringData);

        }

        Console.WriteLine("Disconnecting from server...");

        ns.Close();

        server.Close();

    }

}

**SERVER SIDE CODE:**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class TcpListenerSample

{

    public static void Main()                         //SERVER SIDE

    {

        int recv;

        byte[] data = new byte[1024];

        TcpListener newsock = new TcpListener(9050);

        newsock.Start();

        Console.WriteLine("Waiting for a client...");

        TcpClient client = newsock.AcceptTcpClient();

        NetworkStream ns = client.GetStream();

        string welcome = "Welcome to my test server";

        data = Encoding.ASCII.GetBytes(welcome);

        ns.Write(data, 0, data.Length);

        while (true)

        {

            data = new byte[1024];

            recv = ns.Read(data, 0, data.Length);

            if (recv == 0)

                break;

            Console.WriteLine(

            Encoding.ASCII.GetString(data, 0, recv));

            ns.Write(data, 0, recv);

        }

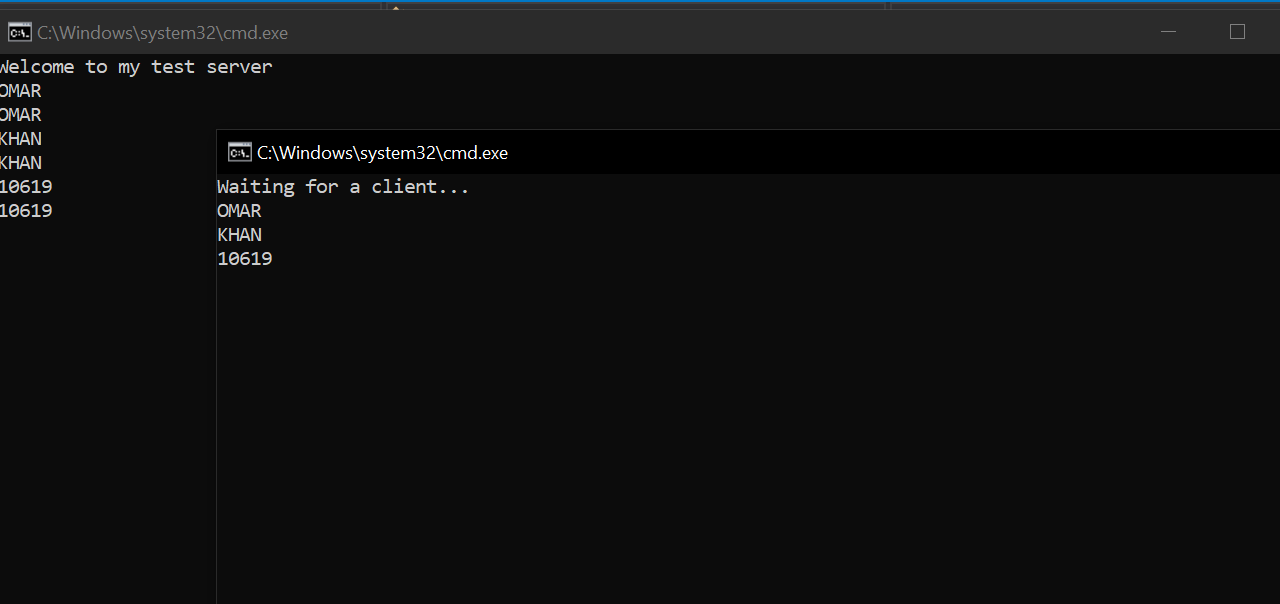
        ns.Close();

        client.Close();

        newsock.Stop();

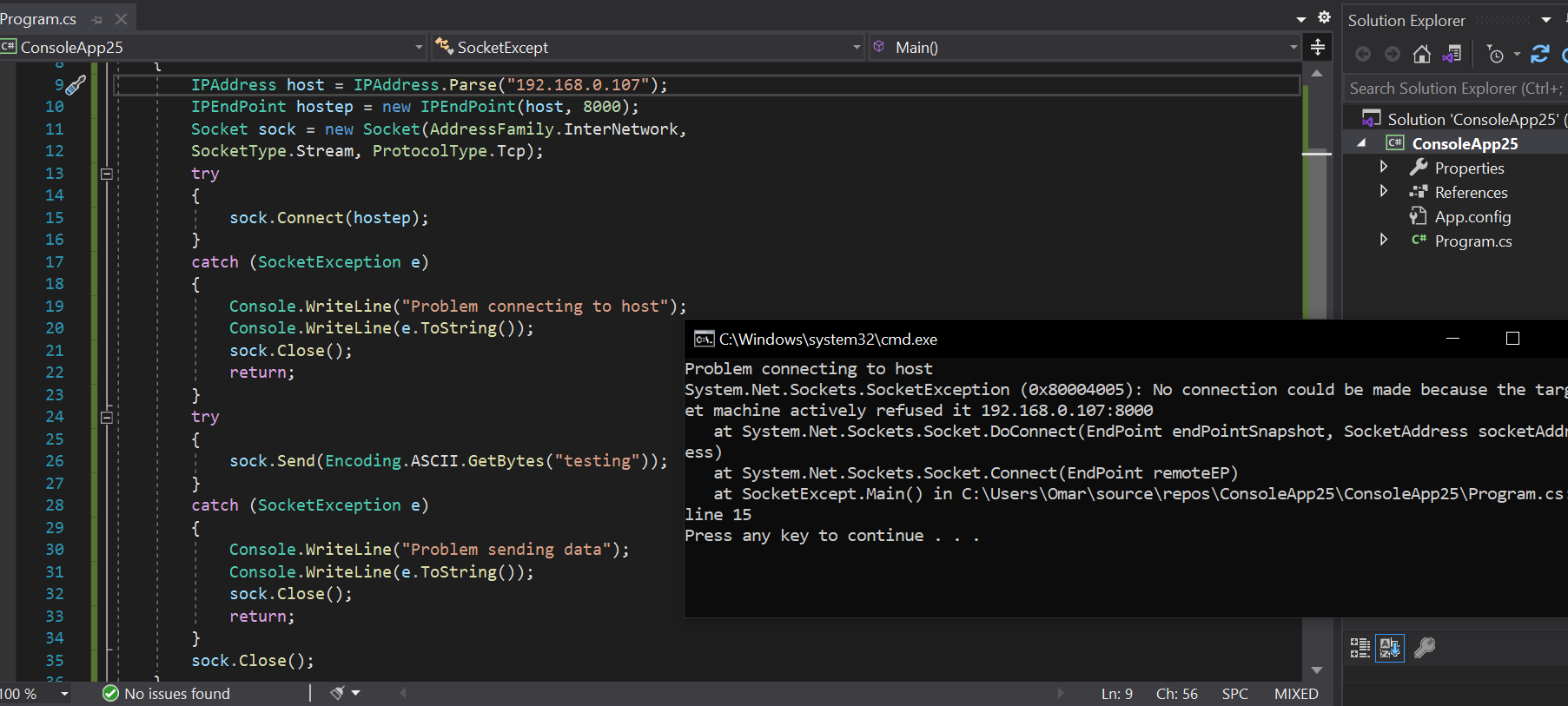
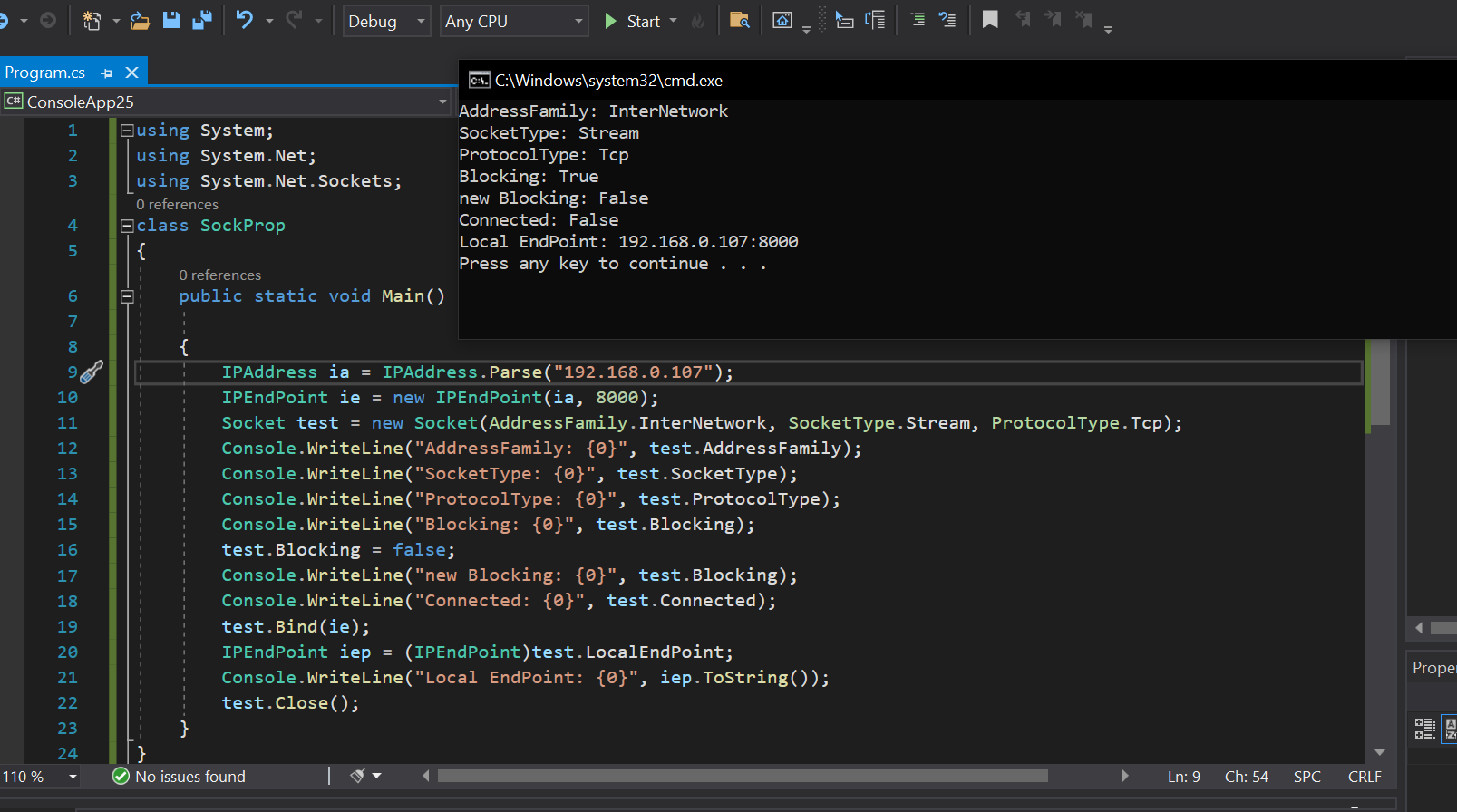
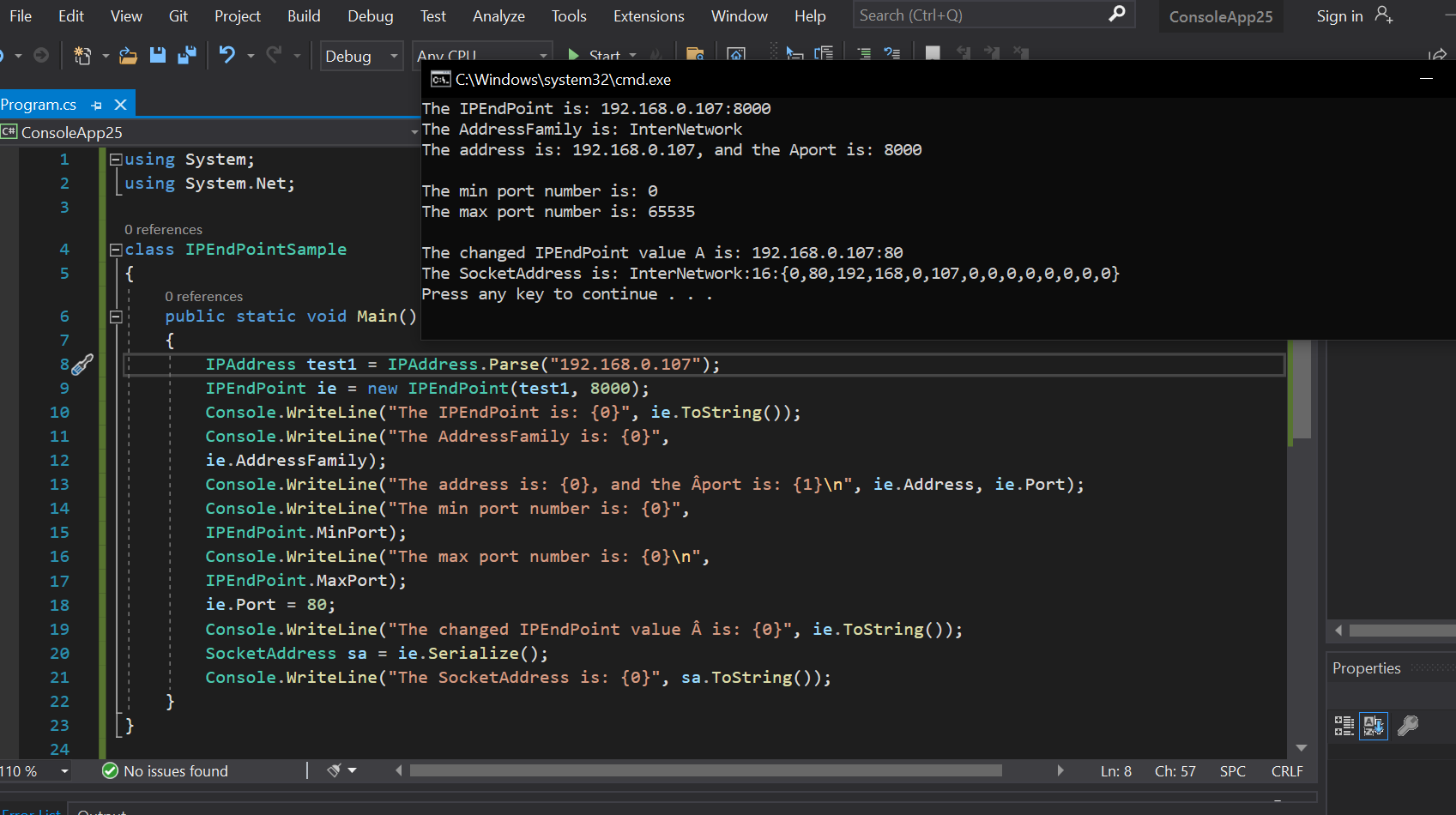
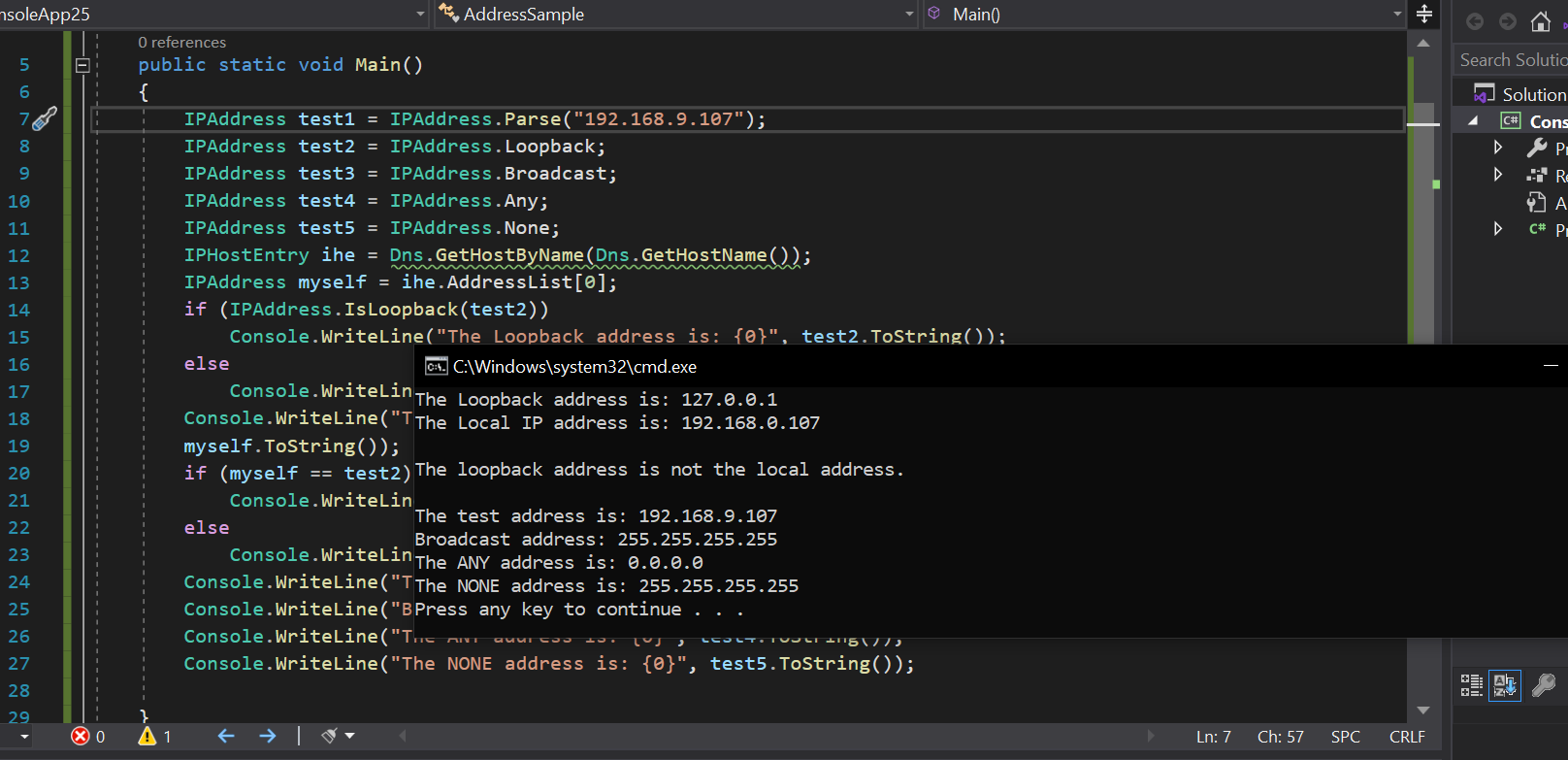
    }

}

**OUTPUT:**

**CHAPTER#03 LISTINGS**

**(3.1-3.4)**

****

LAB#08 (ASYNCHRONOUS)

**SEVER SIDE CODE:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Windows.Forms;

namespace AsynServer

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        private void Form1\_Load(object sender, EventArgs e)

        {

            CheckForIllegalCrossThreadCalls = false;

            TcpListener listener = new TcpListener(IPAddress.Loopback, 11000);

            listener.Start(10);

            listener.BeginAcceptTcpClient(new AsyncCallback(ClientConnect), listener);

        }

        Dictionary<string, TcpClient> lstClients =new Dictionary<string, TcpClient>();

        byte[] b = new byte[1024];

        private void ClientConnect(IAsyncResult ar)

        {

            TcpListener listener =(TcpListener) ar.AsyncState;

            TcpClient client= listener.EndAcceptTcpClient(ar);

            NetworkStream ns = client.GetStream();

            object[] a = new object[2];

            a[0] = ns;

            a[1] = client;

            ns.BeginRead(b, 0, b.Length, new AsyncCallback(ReadMsg), a);

            listener.BeginAcceptTcpClient(new AsyncCallback(ClientConnect), listener);

        }

        private void ReadMsg(IAsyncResult ar)

        {

            object[] a = (object[])ar.AsyncState;

            NetworkStream ns = (NetworkStream) a[0];

            TcpClient client = (TcpClient)a[1];

            int count = ns.EndRead(ar);

            string msg = ASCIIEncoding.ASCII.GetString(b, 0, count);

            if (msg.Contains("@name@"))

            {

                string name = msg.Replace("@name@", "");

                lstClients.Add(name, client);

                lstbxClients.Items.Add(name);

            }

            else

            {

                txtDisplay.Text += msg + Environment.NewLine;

            }

            ns.BeginRead(b, 0, b.Length, new AsyncCallback(ReadMsg), a);

        }

        private void button2\_Click(object sender, EventArgs e)

        {

            TcpClient client = (TcpClient)lstClients[lstbxClients.SelectedItem.ToString()];

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            string textToSend = "Server Says:" + txtMsg.Text;

            sw.WriteLine(textToSend);

            txtDisplay.Text += textToSend + Environment.NewLine;

            sw.Flush();

        }

        private void lstbxClients\_SelectedIndexChanged(object sender, EventArgs e)

        {

        }

    }

}

**CLIENT SIDE CODE:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Windows.Forms;

namespace AysncClient

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        byte[] b = new byte[1024];

        TcpClient client = new TcpClient();

        private void button1\_Click(object sender, EventArgs e)

        {

            CheckForIllegalCrossThreadCalls = false;

            client.Connect(IPAddress.Loopback, 11000);

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            sw.WriteLine("@name@" + txtName.Text);

            sw.Flush();

            ns.BeginRead(b, 0, b.Length, ReadMsg,ns);

        }

        private void ReadMsg(IAsyncResult ar)

        {

            NetworkStream ns =(NetworkStream) ar.AsyncState;

            int count = ns.EndRead(ar);

            txtDisplay.Text += ASCIIEncoding.ASCII.GetString(b, 0, count);

            ns.BeginRead(b, 0, b.Length, ReadMsg, ns);

        }

        private void button2\_Click(object sender, EventArgs e)

        {

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            sw.WriteLine(txtName.Text + " Says: " + txtMsg.Text);

            sw.Flush();

        }

        private void Form1\_Load(object sender, EventArgs e)

        {

        }

    }

}

**ASYNC MULTICLIENT (CLIENT):**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

using System.Net;

using System.Net.Sockets;

using System.IO;

namespace Client

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        byte[] b = new byte[1024];

        TcpClient client = new TcpClient();

        private void Form1\_Load(object sender, EventArgs e)

        {

        }

        private void ReadMsg(IAsyncResult ar)

        {

            NetworkStream ns = (NetworkStream)ar.AsyncState;

            int count = ns.EndRead(ar);

            txtDisplay.Text += ASCIIEncoding.ASCII.GetString(b, 0, count);

            ns.BeginRead(b, 0, b.Length, ReadMsg, ns);

        }

        private void button1\_Click(object sender, EventArgs e)

        {

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            sw.WriteLine(txtName.Text + " Says: " + txtMsg.Text);

            sw.Flush();

        }

        private void button2\_Click(object sender, EventArgs e)

        {

            CheckForIllegalCrossThreadCalls = false;

            client.Connect(IPAddress.Loopback, 11000);

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            sw.WriteLine("@name@" + txtName.Text);

            sw.Flush();

            ns.BeginRead(b, 0, b.Length, ReadMsg, ns);

        }

    }

}

**SERVER:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

using System.Net;

using System.Net.Sockets;

using System.IO;

namespace Server

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        private void Form1\_Load(object sender, EventArgs e)

        {

            CheckForIllegalCrossThreadCalls = false;

            TcpListener listener = new TcpListener(IPAddress.Loopback, 11000);

            listener.Start(10);

            listener.BeginAcceptTcpClient(new AsyncCallback(ClientConnect), listener);

        }

        Dictionary<string,TcpClient> lstClients = new Dictionary<string,TcpClient>();

        byte[] b = new byte[1024];

        private void ClientConnect(IAsyncResult ar)

        {

            TcpListener listener = (TcpListener)ar.AsyncState;

            TcpClient client = listener.EndAcceptTcpClient(ar);

            NetworkStream ns = client.GetStream();

            object[] a = new object[2];

            a[0] = ns;

            a[1] = client;

            ns.BeginRead(b, 0, b.Length, new AsyncCallback(ReadMsg), a);

            listener.BeginAcceptTcpClient(new AsyncCallback(ClientConnect), listener);

        }

        private void ReadMsg(IAsyncResult ar)

        {

            object[] a = (object[])ar.AsyncState;

            NetworkStream ns = (NetworkStream)a[0];

            TcpClient client = (TcpClient)a[1];

            int count = ns.EndRead(ar);

            string msg = ASCIIEncoding.ASCII.GetString(b, 0, count);

            if (msg.Contains("@name@"))

            {

                string name = msg.Replace("@name@", "");

                lstClients.Add(name, client);

                lstbxClients.Items.Add(name);

            }

            else

            {

                txtDisplay.Text += msg + Environment.NewLine;

            }

            ns.BeginRead(b, 0, b.Length, new AsyncCallback(ReadMsg), a);

        }

        private void button1\_Click(object sender, EventArgs e)

        {

            TcpClient client = (TcpClient)lstClients[lstbxClients.SelectedItem.ToString()];

            NetworkStream ns = client.GetStream();

            StreamWriter sw = new StreamWriter(ns);

            string textToSend = "Server Says : " + txtMsg.Text;

            sw.WriteLine(textToSend);

            txtDisplay.Text += textToSend + Environment.NewLine;

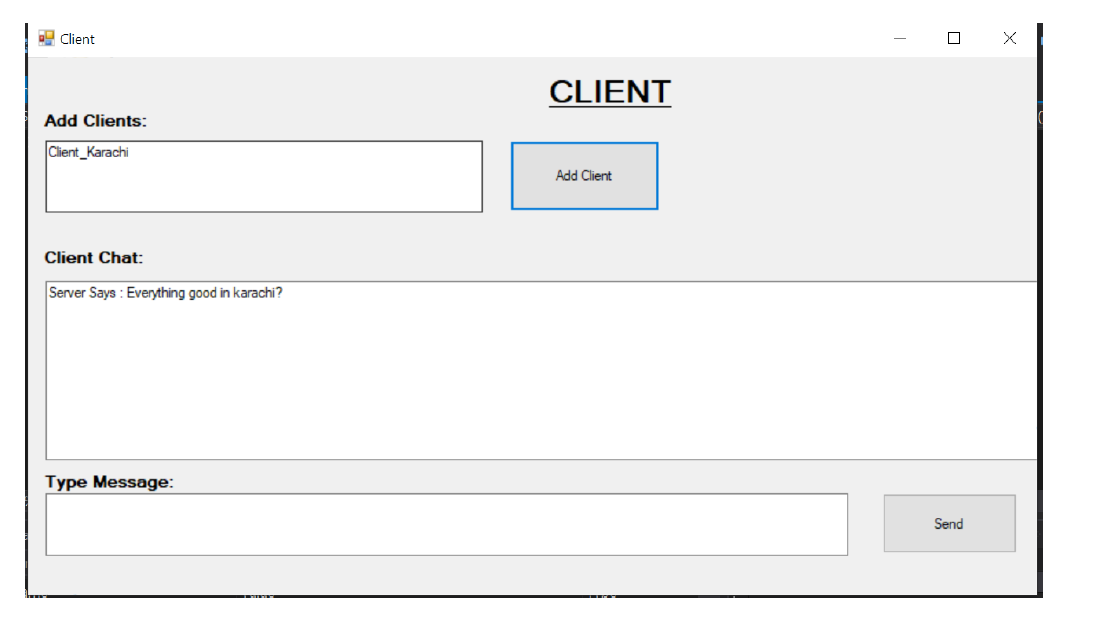
            sw.Flush();

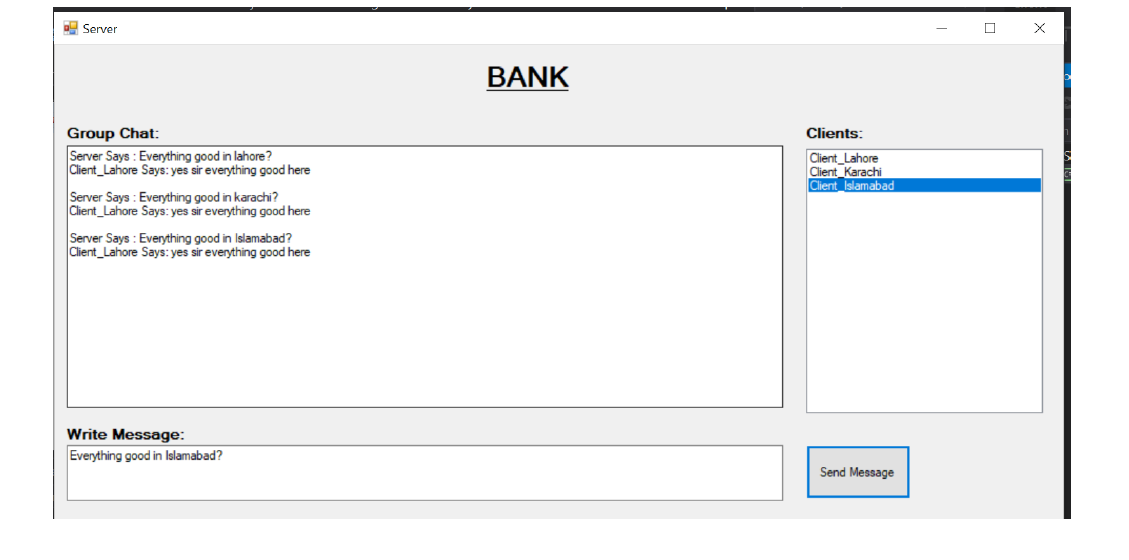
        }

    }

}

**OUTPUT:**

****

****

**SMTP**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Mail;

namespace WindowsFormsApplication9

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

            listBox1.Items.Add("theomyway@gmail.com");

            listBox1.Items.Add("beingfaisalmalik@gmail.com");

        }

        private void button1\_Click(object sender, EventArgs e)

        {

            try

            {

                MailMessage mail = new MailMessage();

                SmtpClient smptpserver = new SmtpClient("smtp.gmail.com");

                mail.From = new MailAddress("theomway@gmail.com");

                string selectedItem = listBox1.Items[listBox1.SelectedIndex].ToString();

                mail.To.Add(selectedItem);

                mail.Subject = textBox1.Text;

                mail.Body = textBox2.Text;

                smptpserver.Port = 587;

                listBox1.SelectionMode = SelectionMode.MultiSimple;

                smptpserver.Credentials = new System.Net.NetworkCredential("theomyway@gmail.com", textBox4.Text);

                Console.WriteLine("theomyway says:");

                string msg = textBox2.Text;

                listBox2.Items.Add("theomyway@gmail.com to:" + selectedItem + ":" + msg);

                smptpserver.EnableSsl = true;

                smptpserver.Send(mail);

                MessageBox.Show("SUCCESSFULLY SENT!");

            }

            catch (Exception ex)

            {

                MessageBox.Show(ex.ToString());

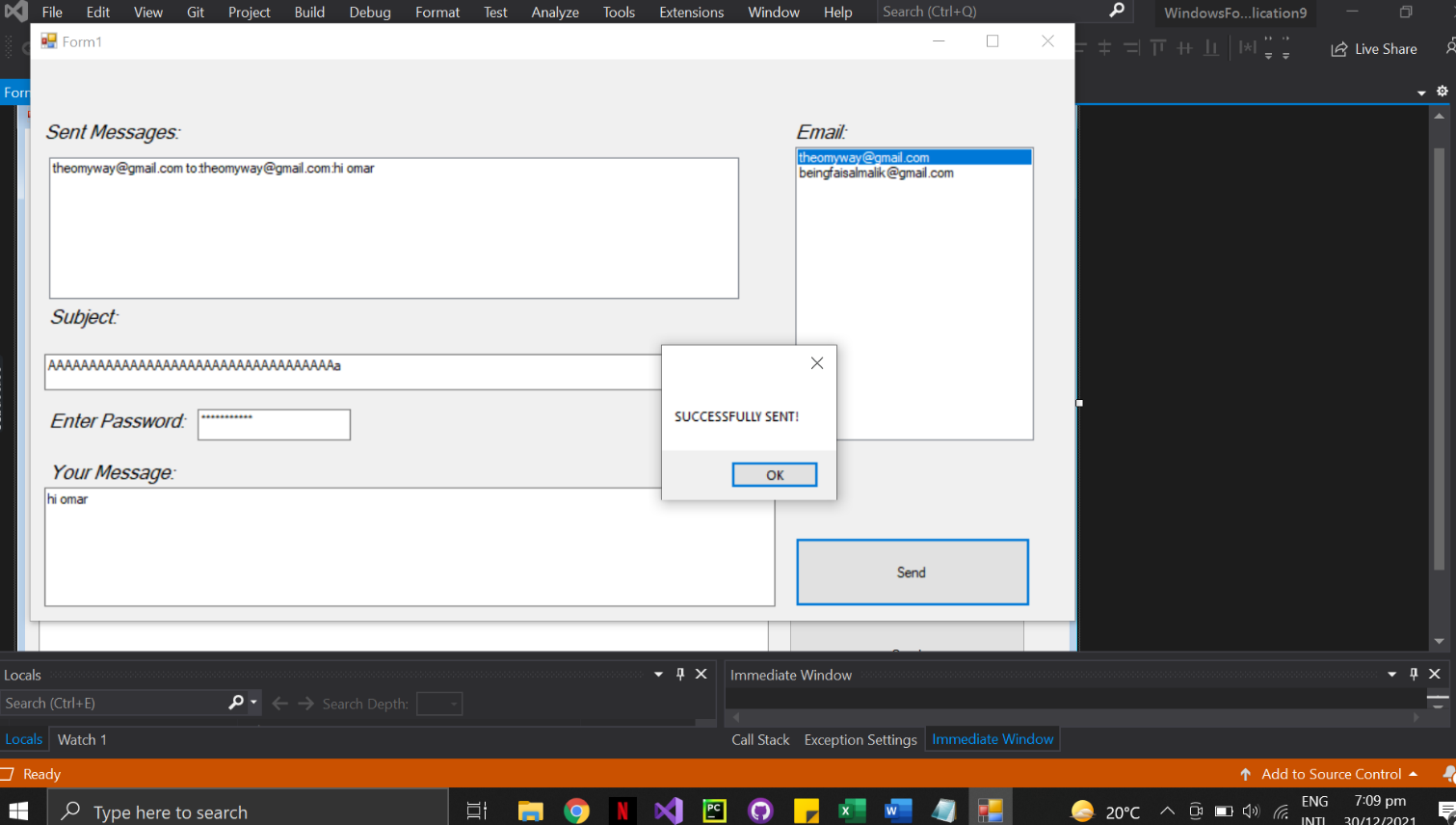
            }

        }

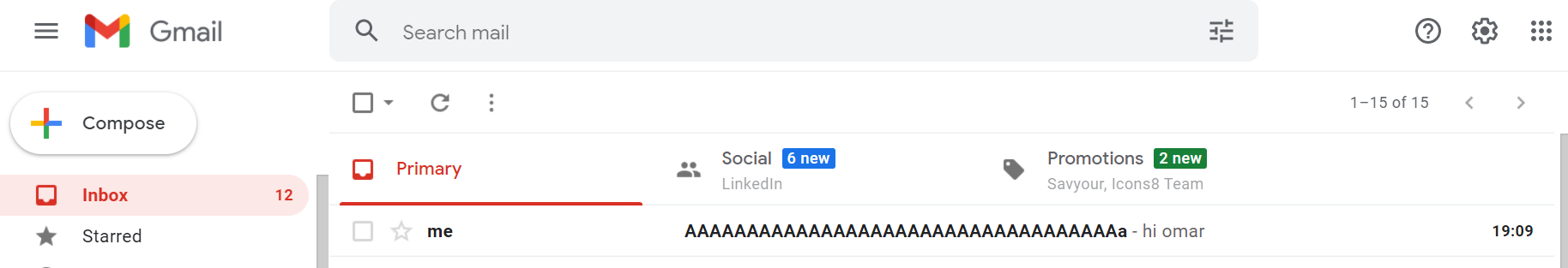
    }

}

**OUTPUT:**

****

**OUTPUT:**



**FTP:**

using System.Web;

using System.Web.Optimization;

namespace ftpweb

{

    public class BundleConfig

    {

        // For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862

        public static void RegisterBundles(BundleCollection bundles)

        {

            bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

                        "~/Scripts/jquery-{version}.js"));

            bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

                        "~/Scripts/jquery.validate\*"));

            // Use the development version of Modernizr to develop with and learn from. Then, when you're

            // ready for production, use the build tool at https://modernizr.com to pick only the tests you need.

            bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

                        "~/Scripts/modernizr-\*"));

            bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

                      "~/Scripts/bootstrap.js"));

            bundles.Add(new StyleBundle("~/Content/css").Include(

                      "~/Content/bootstrap-add.css",

                      "~/Content/site.css"));

        }

    }

}

using System.Web;

using System.Web.Mvc;

namespace ftpweb

{

    public class FilterConfig

    {

        public static void RegisterGlobalFilters(GlobalFilterCollection filters)

        {

            filters.Add(new HandleErrorAttribute());

        }

    }

}

using ftpweb.Models;

using System.IO;

using System.Web.Mvc;

namespace ftpweb.Controllers

{

    public class HomeController : Controller

    {

        public ActionResult Index()

        {

            return View();

        }

        [HttpPost]

        public ActionResult Index(DocumentUploaderModel DocumentUpload)

        {

            string Direc = Server.MapPath("~/Content/Document");

            string Documentname = Path.GetFileName(DocumentUpload.File.FileName);

            string CompleteDirectory = Path.Combine(Direc, Documentname);

            DocumentUpload.File.SaveAs(CompleteDirectory);

            return View();

        }

        public FileResult Download()

        {

            string Direc = Server.MapPath("~/Content/Document");

            string Documentname = Path.GetFileName("Capture.jpg");

            string CompleteDirectory = Path.Combine(Direc, Documentname);

            return File(CompleteDirectory, "image/jpg", "DownloadedFile.jpg");

        }

        public ActionResult About()

        {

            ViewBag.Message = "Your application description page.";

            return View();

        }

        public ActionResult Contact()

        {

            ViewBag.Message = "Your contact page.";

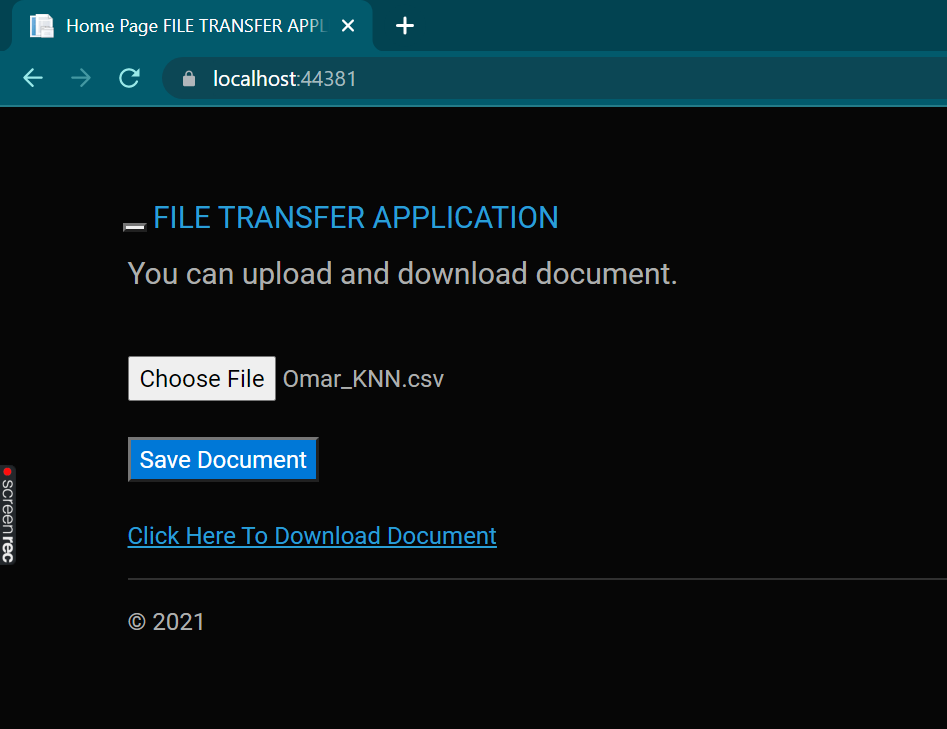
            return View();

        }

    }

}

**OUTPUT:**

****

**VOICE CALL:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.IO.Ports;

namespace WindowsApplication1

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        private void button1\_Click(object sender, EventArgs e)

        {

            SerialPort sp = new SerialPort();

            sp.PortName = "COM10";

            sp.BaudRate = 9600;

            sp.Parity = Parity.None;

            sp.DataBits = 8;

            sp.StopBits = StopBits.One;

            sp.Handshake = Handshake.XOnXOff;

            sp.DtrEnable = true;

            sp.RtsEnable = true;

            sp.Open();

            if (!sp.IsOpen)

            {

                MessageBox.Show("Serial port is not opened");

                return;

            }

            sp.WriteLine("AT" + Environment.NewLine);

            sp.WriteLine("ATD=\"" + "Destination Number;" + "\"" + Environment.NewLine);

        }

    }

}