Лабораторная работа №3

Работа с сокетами TCP в .NET

Целью этой лабораторной является создание двух приложения. Первое – сервер, реализующий комнату чата для обмена текстовыми сообщениями. Второе – клиент для работы с чатом.

Серверное приложение должно выполнять следующие функции:

- принимать входящие соединения;
- вести список пользователей;
- передавать пользователям информацию о новых участниках, их сообщениях и информацию об уходящих участниках.

Клиентское приложение должно:

- подключаться к серверу;
- передавать серверу имя пользователя, его сообщения и сигнализировать о выходе из чата;
- получать от сервера информацию о действиях других участников чата.

Сервер и клиент должны обмениваться короткими текстовыми сообщениями разных типов. Ниже приводится синтаксис и семантика протокола их взаимодействия.

Клиент может посылать серверу следующие команды:

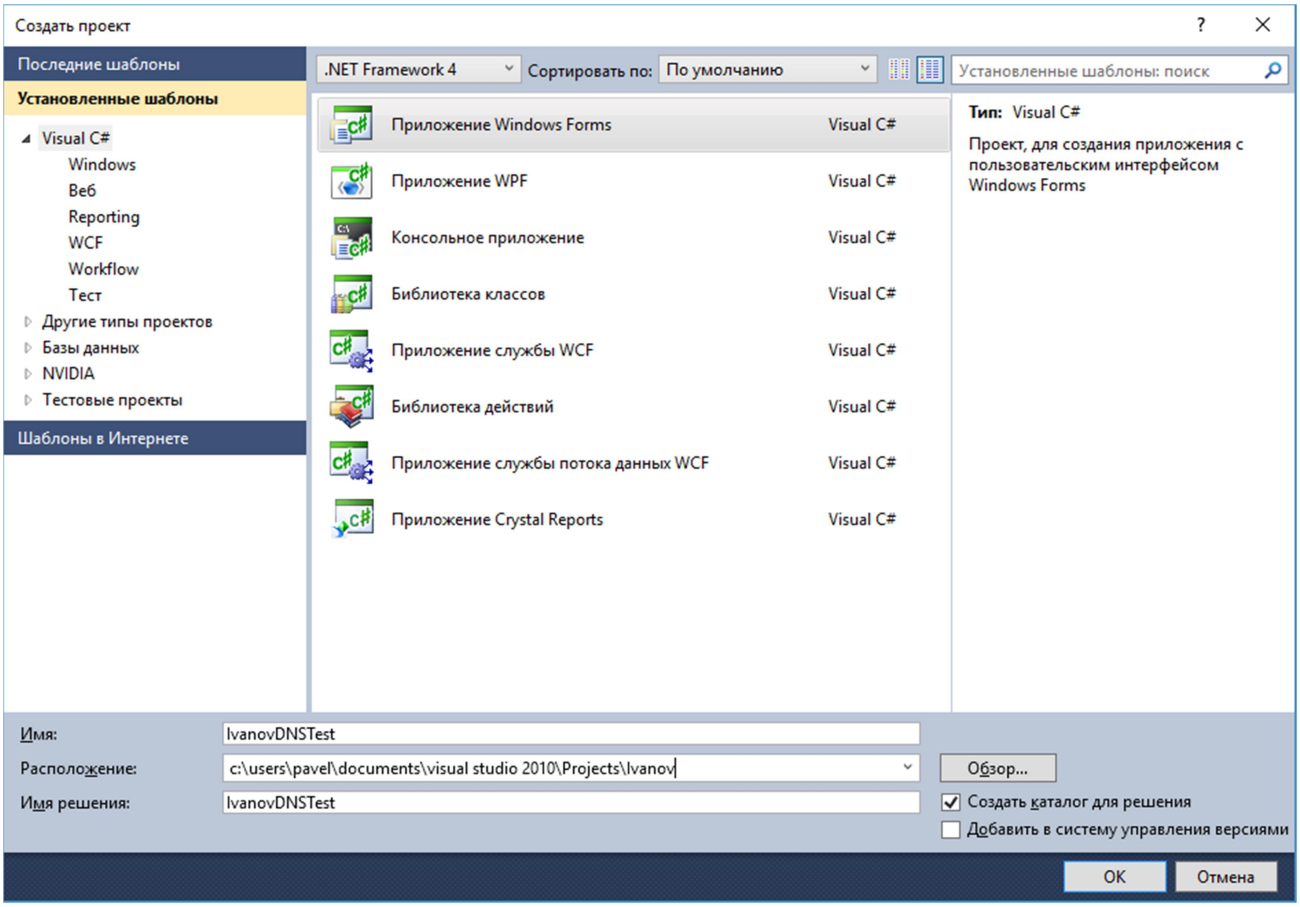
- **name <имя>** - задать для пользователя имя, где <имя> - произвольные текст;
- **message <сообщение>** - написать сообщение в чат;
- **quit** – пользователь покидает чат.

После получения команды от клиента сервер выполняет нужное действие и всем остальным участникам чата (кроме того, кто инициировал событие) высылает информационные сообщения:

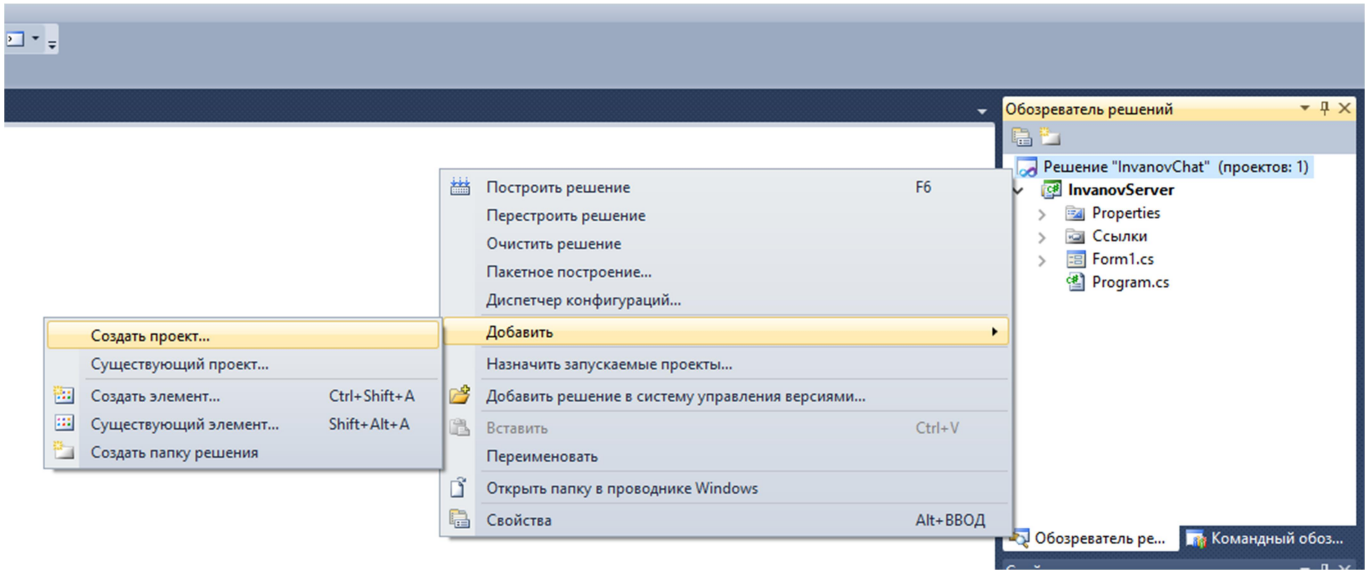
- **new <имя>** - в чате новый участник <имя>;
- **message <имя: сообщение>** - новое сообщения от участника;
- **exit <имя>** – пользователь покидает чат.

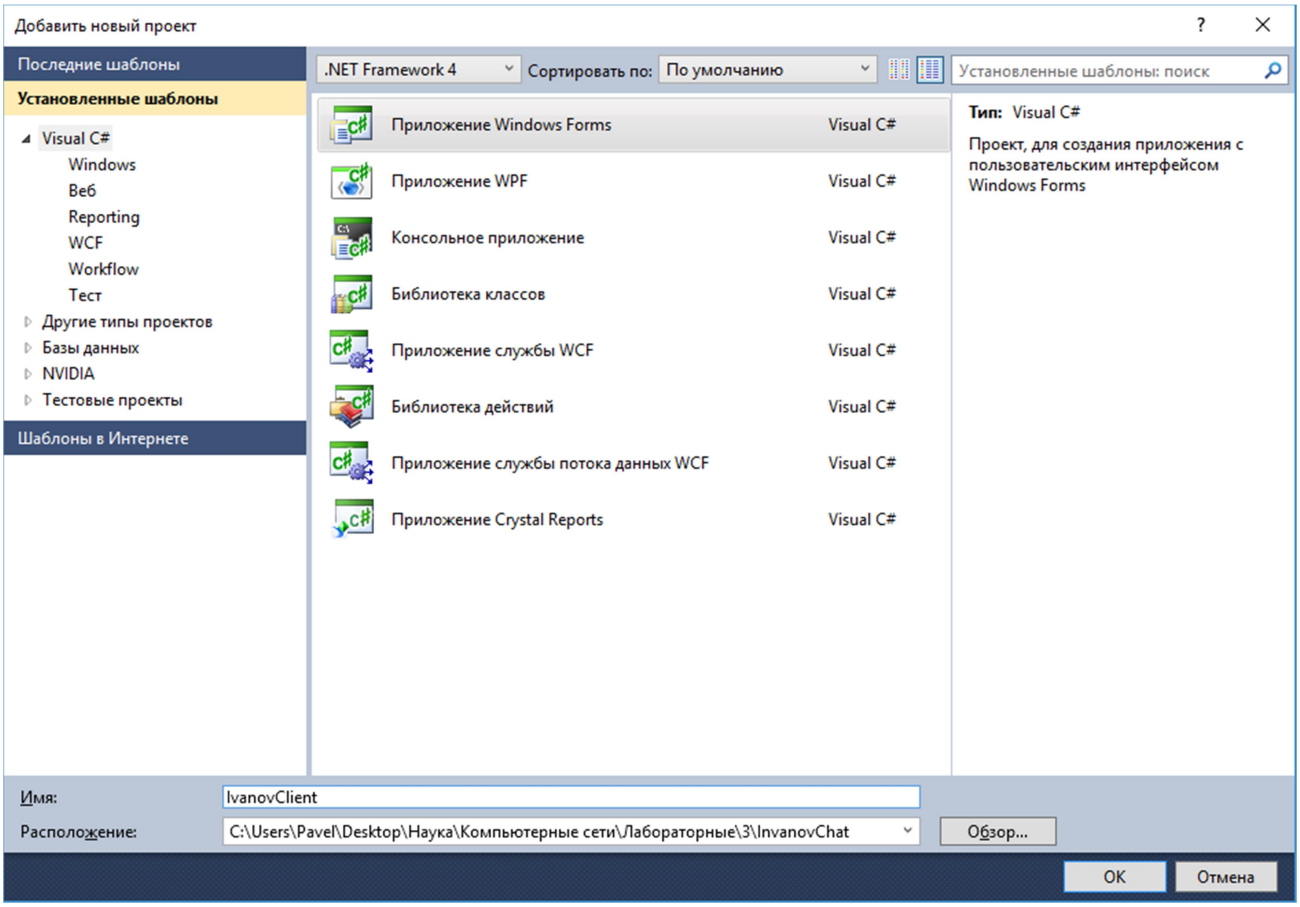
Шаг 1. Открываем Microsoft Visual Studio и создаем 2 новых проекта C#/Windows Forms

В Visual Studio совокупность нескольких проектов называется решением. Сначала создадим решение и проект для сервера, как это делалось в предыдущих лабораторных работах. Например, проекту дадим название IvanovServer, а всему решению имя IvanovChat.

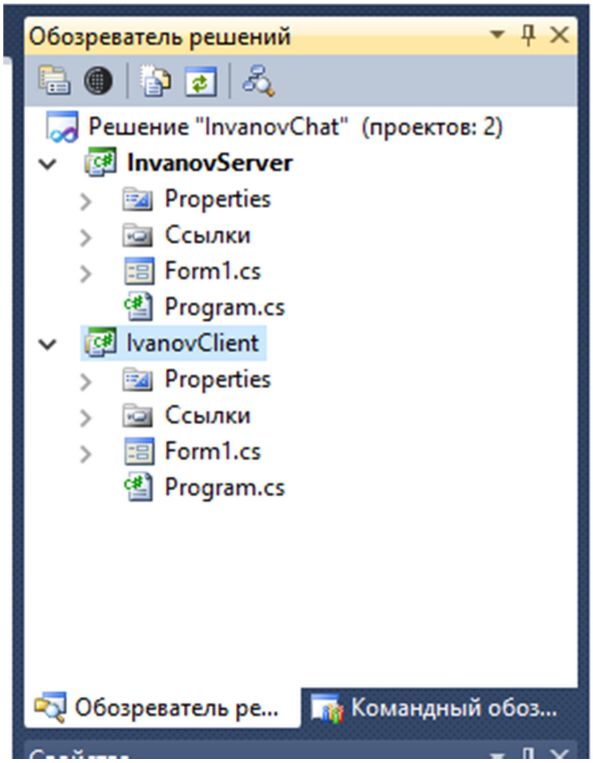


Далее сразу создадим внутри этого решения второй проект IvanovClient. Для этого надо открыть обозреватель решений, правой кнопкой мыши щелкнуть на строке с решением и в меню выбрать команду Добавить/Создать проект.

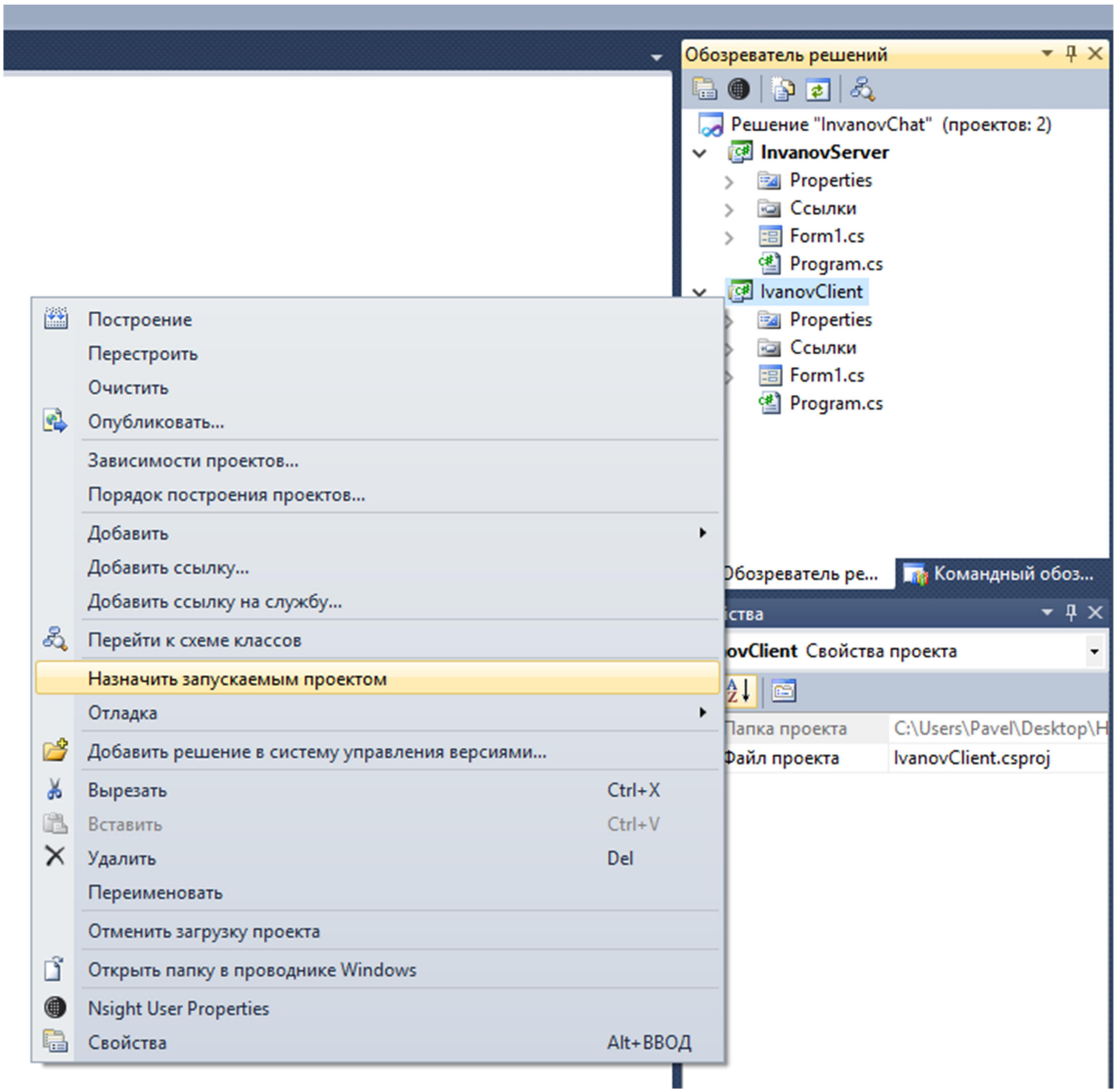




Если все будет сделано правильно, то в обозревателе решения можно будет увидеть оба проекта.



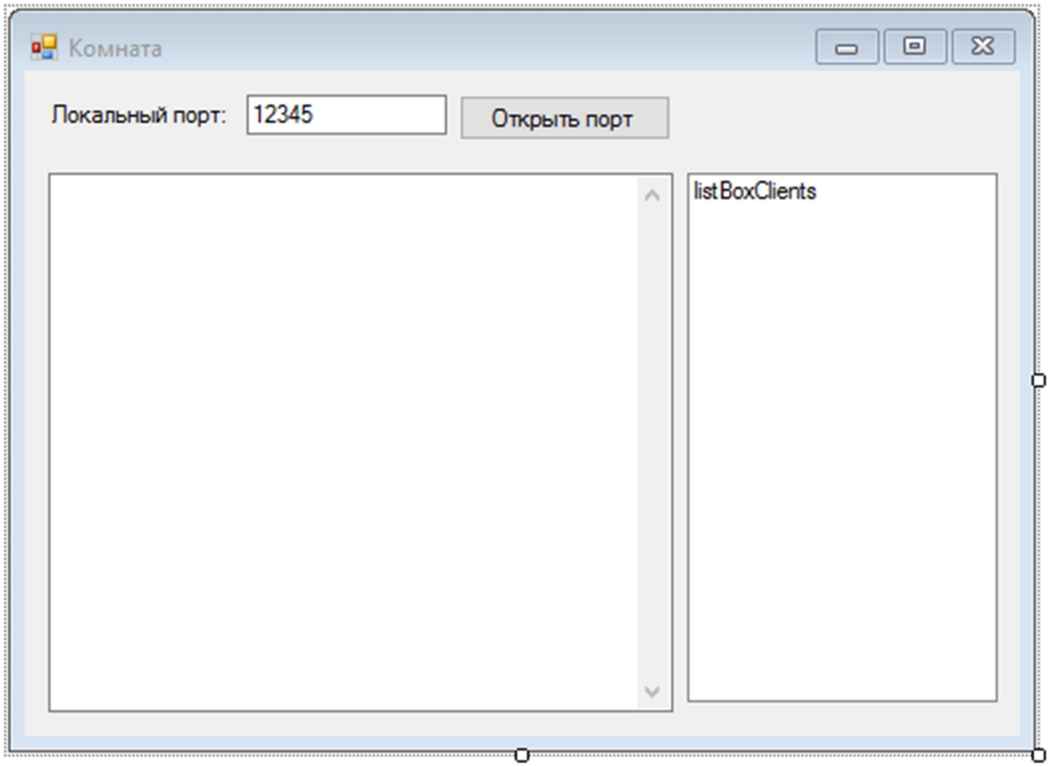
Один из проектов является активным, а другие пассивны. Чтобы сменить активный проект надо в обозревателе решений щелкнуть правой кнопкой на строке с проектом и в меню выбрать пункт «Назначить запускаемым проектом».



Шаг 2.1 Создание визуального интерфейса серверного приложения

На рисунке показан визуальный интерфейс, который необходимо создать:

- текстовые поля **textBoxLocalPort** (локальный порт) и **textBoxLog** (поле текстового вывода);
- кнопка **buttonBind** (открыть порт);
- список **listBoxClients** (класс ListBox) для отображения списка пользователей;
- таймер **timer1** (класс Timer).



Шаг 2.2 Создание исходного кода сервера. Подготовка переменных для хранения данных

Вначале не забудьте, как и ранее, подключить к файлу исходного кода формы две библиотеки: System.Net и System.Net.Sockets.

Серверному приложению понадобиться хранить информацию о текущих клиентах. Во-первых, для каждого клиента надо хранить информацию о сокете, через который с ним происходит общение. Кроме того, нужно хранить имя, которым решил назваться пользователь. Для этого в программе создадим класс **ClientInfo**. В этом классе укажем два поля: **socket** типа **Socket** и **name** типа **string**, как это показано на иллюстрации. Этот класс содержит один метод **ToString**, который возвращает текст в виде: <имя> (<адрес>: <порт>). Этот метод нужен для адекватного отображения перечня клиентов в поле-списке **listBoxClients**.

Для отслеживания входящих соединений в классе формы объявим переменную **listener** типа **TcpListener**. Этот класс является удобной «надстройкой» над классом Socket, упрощающей написание кода.

Для хранения списка пользователей также объявим переменную **clients** типа **List<ClientInfo>**. Такое объявление гласит, что в переменной clients будет храниться список объектов типа ClientInfo.

```
public partial class Form1 : Form
{

    class ClientInfo //Класс для хранения информации о клиенте
    {
        public Socket socket;//Сокет
        public string name;//Имя пользователя

        public override string ToString() //метод для преобразования объекта в текстовую строку
        {
            return name + " (" + socket.RemoteEndPoint + ")";
        }
    }

    TcpListener listener; //Объект для приема входящих TCP-соединений
    List<ClientInfo> clients; //список информации о пользователях

    public Form1()
    {
        InitializeComponent();
    }
```

Шаг 2.3 Создание исходного кода сервера. Создание сокета входящих соединений

Для кнопки **buttonBind** создадим обработчик события **Click**. Этот обработчик будет выполнять следующие действия:

- создавать объект **TcpListener** и связывать его с портом, указанным в поле **textBoxLocalPort**;
- начинать прослушивание входящих соединений;
- создавать пустой список **clients** для хранения данных о клиентах;
- включать таймер для обработки соединений

```
private void buttonBind_Click(object sender, EventArgs e)
{
    try
    {
        //преобразуем текст из textBoxLocalPort в число localPort
        int localPort = Int32.Parse(textBoxLocalPort.Text);

        //создаем пару (адрес, порт) для открытия сокета
        IPEndPoint localPoint = new IPEndPoint(IPAddress.Any, localPort);

        //создаем объект TcpListener из сохраняем его в переменную listener
        listener = new TcpListener(localPoint);

        //запускаем процесс прослушивания сокета
        listener.Start();

        //создаем список пользователей (изначально пустой)
        clients = new List<ClientInfo>();

        //включаем таймер для периодической проверки
        timer1.Enabled = true;

        //выводим текст об успешном открытии сокета
        textBoxLog.AppendText("Открыт TCP порт " + textBoxLocalPort.Text + "\n");
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }
}
```

Шаг 2.4 Создание исходного кода сервера. Обработка соединений

Для таймера **timer1** создадим обработчик события **Tick**. Этот обработчик выполняет следующие действия:

- проверять наличие новых соединений, вызывая метод **CheckListener**;
- в цикле от последнего к первому элементу списка **clients** последовательно проверять сокеты на наличие новых данных
- если данные имеются, то будет происходить их получение и преобразование в текст;
- после этого будет вызываться метод **DoClient**, который обрабатывает полученное сообщение.

```
private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        CheckListener();//проверяем новые подключения

        //для всех текущих подключений проверяем сокет на новые данные
        for (int i = clients.Count - 1; i >= 0; i--)
        {
            //выбираем i-й элемент списка и запоминаем в перемнной client
            ClientInfo client = clients[i];

            if (client.socket.Available > 0) //если есть новые данные
            {
                //получаем данные и преобразуем в текст
                byte[] data = new byte[client.socket.Available];
                int data_size = client.socket.Receive(data);
                string text_data = Encoding.UTF8.GetString(data, 0, data_size);
                //вызываем обработку
                DoClient(client,text_data);
            }
        }
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }

}
```

Для завершения исходного кода сервера необходимо написать методы **CheckListener** и **DoClient**. Код этих методов можно разместить где угодно внутри класса (но не внутри других методов), например, сразу за обработчиком **timer1_Tick.**

Шаг 2.5 Создание исходного кода сервера. Прием входящих соединений

Ниже приведен исходный код метода **CheckListener**.

```
        //метод проверки новых соединений
        private void CheckListener()
        {
            if (listener.Pending()) //есть новые соеденения!
            {
                //создаем объект ClientInfo для хранения информации о новом пользователе
                ClientInfo newClient = new ClientInfo();
                //создаем сокет для нового пользователя и сохраняем его в поле newClient.socket
                newClient.socket = listener.AcceptSocket();
                //добавляем объект newClient в список clients
                clients.Add(newClient);
                //выводим информацию
                textBoxLog.AppendText("Пользователь " + newClient.socket.RemoteEndPoint + " подключился\n");
            }
        }
```

Этот метод выполняет следующие действия:

- проверяет наличие новых соединений, вызывая метод **listener.Pending**;
- в случае их наличия создает новый объект **ClientInfo** для хранения информации о новом соединении;
- создает сокет методом **listener.AcceptSocket** (принять соединение);
- добавляет в список client информацию о новом клиенте.

Шаг 2.6 Создание исходного кода сервера. Обработка сообщений от клиента

Ниже приведен исходный код метода **DoClient**.

```
//метод обработки данных text_data от клиента client
private void DoClient(ClientInfo client, string text_data)
{
    if (text_data.StartsWith("name ")) //текст начаниается с "name "
    {
        //запоминаем имя пользователя в поле client.name
        client.name = text_data.Substring(5);
        //добавляем пользователя в список listBoxClients
        listBoxClients.Items.Add(client);
        //оптравляем пользователям информацию о новом пользователе
        SendToClients("new " + client.name, client);
        //выводим информацию
        textBoxLog.AppendText("Пользователь " + client.socket.RemoteEndPoint + " выбрал имя " + client.name + "\n");
    }

    if (text_data == "quit") //текст "quit"
    {
        //оптравляем пользователям информацию об уходящем пользователе
        SendToClients("exit " + client.name, client);
        //выводим информацию
        textBoxLog.AppendText("Пользователь " + client.socket.RemoteEndPoint + " покинул комнату\n");
        //закрываем соединение
        client.socket.Shutdown(SocketShutdown.Both);
        client.socket.Close();
        //убираем клиента из списка listBoxClients
        listBoxClients.Items.Remove(client);
        //убираем клиента из списка clients
        clients.Remove(client);
    }

    if (text_data.StartsWith("message ")) //текст начаниается с "message "
    {
        //выделяем из текста сообщение
        string message = text_data.Substring(8);
        //оптравляем пользователям сообщение
        SendToClients("message " + client.name + ": " + message, client);
        //выводим информацию
        textBoxLog.AppendText(client.name+": "+message + "\n");
    }

}
```

Этот метод имеет достаточно большой код, разберем его. Метод состоит из трех секций. Каждая из них проверяет, является ли сообщение одной из команд: **name**, **message** или **quit**. Для проверки используется метод **text_data.StartsWith**, который проверяет, начинается ли строка text_data с указанной последовательности символов.

Для команды **name** сервер выполняет следующие действия:

- сохраняет в поле **client.name** полученное имя, используя метод **text_data.Substring**, которая выдает все символы в строке **text_data** начиная с указанного номера и до конца;
- добавляет в поле **listBoxClients** новый элемент;
- с помощью метода **SendToClients** посылает всем клиентам сообщение **new**.

Для команды **quit** сервер выполняет следующие действия:

- с помощью метода **SendToClients** посылает всем клиентам сообщение **exit**;
- разрывает соединение;
- удаляет элемент client из поля **listBoxClients** и списка clients.

И наконец, для команды **message** сервер с помощью метода **SendToClients** посылает всем клиентам сообщение **message**.

Для рассылки сообщений клиентами запускается метод **SendToClients**, приведенный ниже. Этот метод высылает команду **command** всем клиентам, но не клиенту, указанному в параметре **exceptOf**. Метод можно разместить сразу под методом **DoClient**.

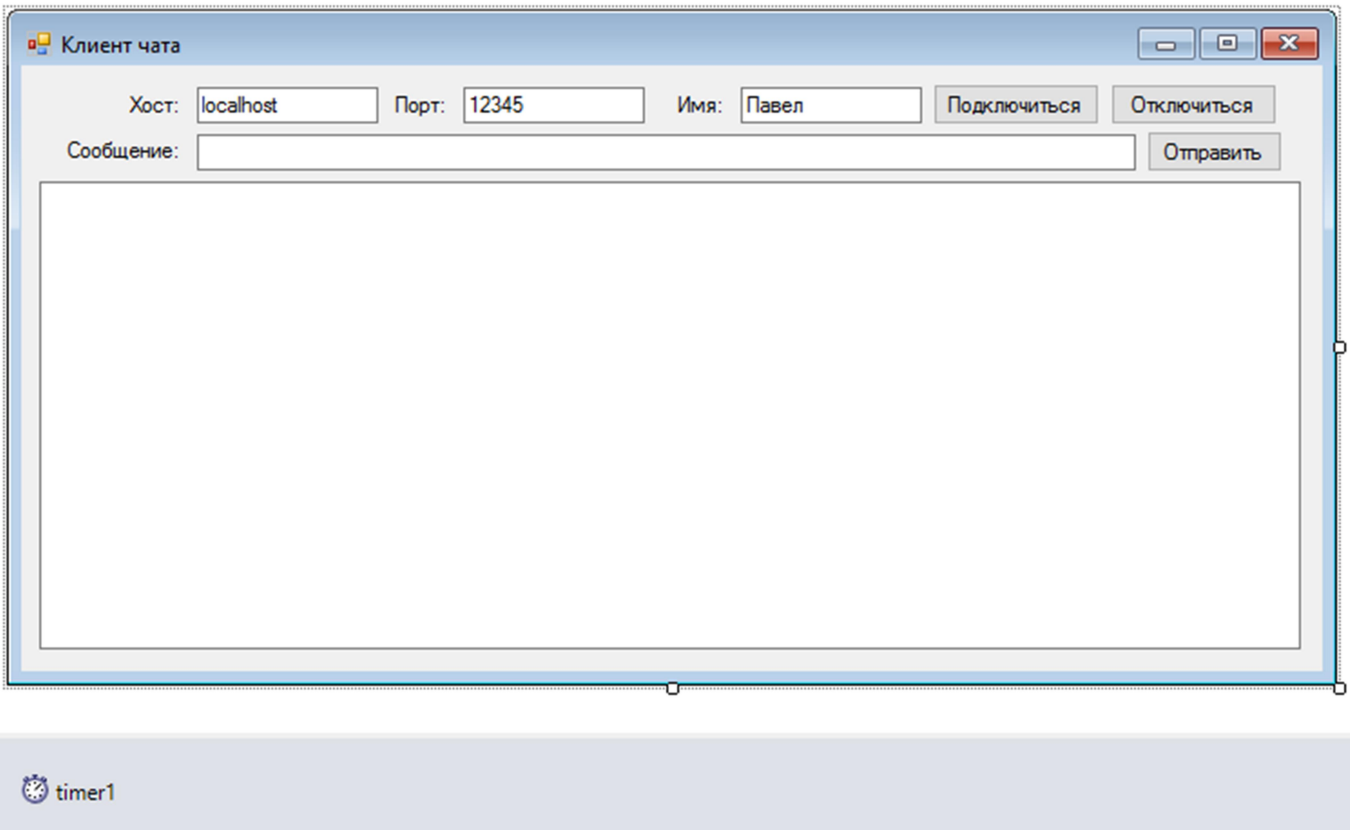
```
//отправка данных command всем клиентам кроме клиента exceptOf
private void SendToClients(string command, ClientInfo exceptOf)
{
    //для всех клиентов из списка clients с первого по последний
    for (int i = 0; i < clients.Count; i++)
    {
        ClientInfo client = clients[i];
        if (client != exceptOf)
        {
            try
            {
                //преобразуем текст в байты
                byte[] data = Encoding.UTF8.GetBytes(command);
                //отправляем
                client.socket.Send(data);
            }
            catch (Exception exc)
            {
                textBoxLog.AppendText(exc.Message + "\n");
            }

        }
    }
}
```

Шаг 3.1 Создание визуального интерфейса клиентского приложения

Сервер закончен, поздравляю! Теперь время переключиться на клиентское приложение, которое будет во многом похоже на приложение из лабораторной работы №2.

На рисунке показан визуальный интерфейс, который необходимо создать:

- текстовые поля **textBoxHost** (хост), **textBoxPort** (порт), **textBoxName** (имя), **textBoxMessage** (сообщение) и **textBoxLog** (поле текстового вывода);
- кнопки **buttonConnect** (подключиться), **buttonDisconnect** (отключиться) и **buttonSend** (отправить); для кнопки «Отключиться» задайте свойство **Enabled** (доступно) равным false.
- таймер **timer1** (класс Timer).



Шаг 3.2 Создание исходного кода клиента

Вначале не забудьте, как и ранее, подключить к файлу исходного кода формы две библиотеки: System.Net и System.Net.Sockets.

В классе формы объявим переменную **socket** для хранения информации о сокете. Также создадим вспомогательный метод **SendToServer**, который будет отправлять строку **command**.

```
public partial class Form1 : Form
{
    Socket socket;

    private void SendToServer(string command)
    {
        //переводим комманду command в список байт
        byte[] data = Encoding.UTF8.GetBytes(command);
        //отправляем данные
        socket.Send(data);
    }

    public Form1()
    {
        InitializeComponent();
    }
```

Для кнопки **buttonConnect** создадим обработчик события **Click**. Этот обработчик будет выполнять следующие действия:

- создавать объект **Socket**, настроенный на протокол TCP;
- подключаться к серверу по данным из **textBoxHots** и **textBoxPort**;
- отправлять серверу команду **name** с данными из **textBoxName**;
- включить таймер для обработки входящих данных и управлять кнопками Connect/Disconnect.

```
private void buttonConnect_Click(object sender, EventArgs e)
{
    try
    {
        //создаем объект Socket и сохраняем его в переменную socket
        //в качестве параметров указываем: сеть интернет, передача данных в виде потока байт, протокол TCP
        socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        //подключаемся к серверу
        socket.Connect(textBoxHost.Text, Int32.Parse(textBoxPort.Text));

        //передаем серверу имя пользователя
        SendToServer("name " + textBoxName.Text);

        //включаем таймер для периодической проверки входящих сообщений
        timer1.Enabled = true;

        //выводим текст об успешном открытии сокета
        textBoxLog.AppendText("Подключено к " + textBoxHost.Text+":"+textBoxPort.Text + "\n");

        //меняем доступность кнопок "Подключиться и отключиться"
        buttonConnect.Enabled = false;
        buttonDisconnect.Enabled = true;
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }
}
```

Для кнопки **buttonDisconnect** создадим обработчик события **Click**. Этот обработчик выполняет следующие действия:

- посылает серверу команду **quit**;
- закрывает соединение;
- останавливает таймер и меняет состояние кнопок.

```
private void buttonDisconnect_Click(object sender, EventArgs e)
{
    try
    {
        //посылаем серверу команду о выходе из чата
        SendToServer("quit");
        //закрываем получение и отправку данных
        socket.Shutdown(SocketShutdown.Both);
        //закрываем соект
        socket.Close();
        //отключаем таймер
        timer1.Enabled = false;
        //меняем доступность кнопок
        buttonConnect.Enabled = true;
        buttonDisconnect.Enabled = false;
        //выводим сообщение об отключении
        textBoxLog.AppendText("Отключено\n");
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }
}
```

Ниже приведен код обработчика кнопки **buttonSend**. Он просто отправляет серверу команду **message**.

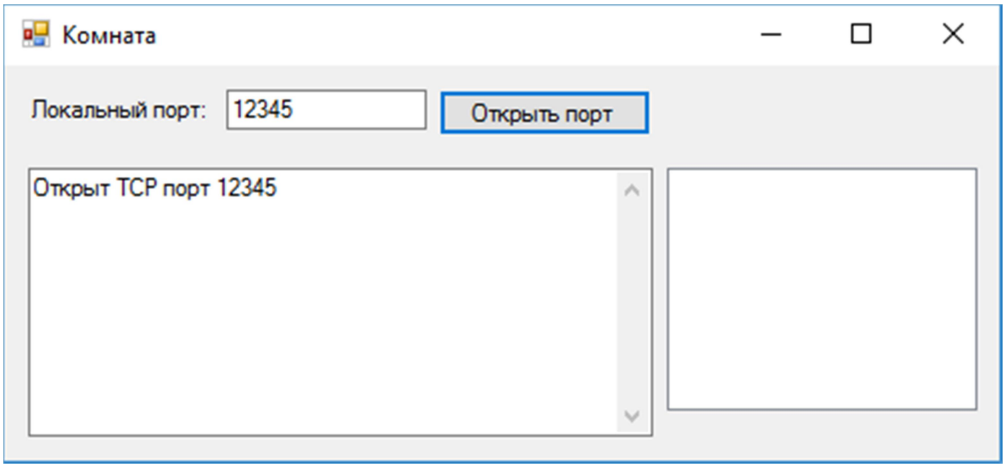
```
private void buttonSend_Click(object sender, EventArgs e)
{
    try
    {
        //посылваем серверу сообщение
        SendToServer("message " + textBoxMessage.Text);
        //выводим информацию
        textBoxLog.AppendText(textBoxName.Text + ": " + textBoxMessage.Text+"\n");
    }
    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }

}
```

Последний и наиболее сложный обработчик – это обработчик таймера **timer1**. Этот обработчик очень похож на метод **DoClient** у сервера: также сначала определяет тип полученного сообщения и в зависимости от этого выводит на экран разную информацию:

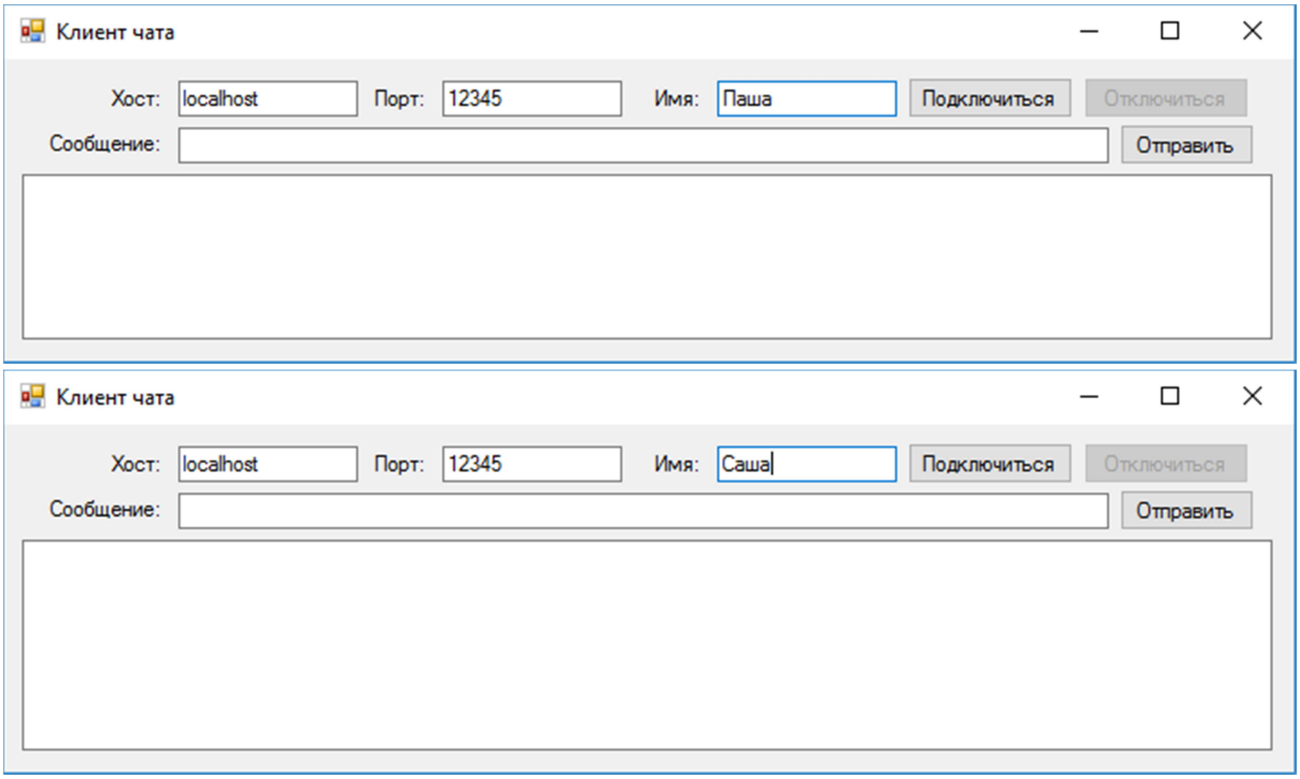
```
private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        if (socket.Available > 0) //если есть новые данные
        {
            //создаем буффер для получения данных
            byte[] data = new byte[socket.Available];
            //получаем данные
            int data_size = socket.Receive(data);
            //преобразуем данные в текст
            string text_data = Encoding.UTF8.GetString(data, 0, data_size);

            if (text_data.StartsWith("new ")) //текст начинается с "new "
            {
                //выводим сообщение о новом пользователе
                textBoxLog.AppendText(text_data.Substring(4) + " вошел в чат\n");
            }

            if (text_data.StartsWith("exit ")) //текст начинается с "exit "
            {
                //выводим сообщение об уходящем пользователе
                textBoxLog.AppendText(text_data.Substring(5) + " покинул чат\n");
            }

            if (text_data.StartsWith("message ")) //текст начинается с "message "
            {
                //выводим реплику другого пользователя
                textBoxLog.AppendText(text_data.Substring(8) + "\n");
            }
        }
    }

    catch (Exception exc)
    {
        textBoxLog.AppendText(exc.Message + "\n");
    }
}
```

Шаг 4. Тестирование программы

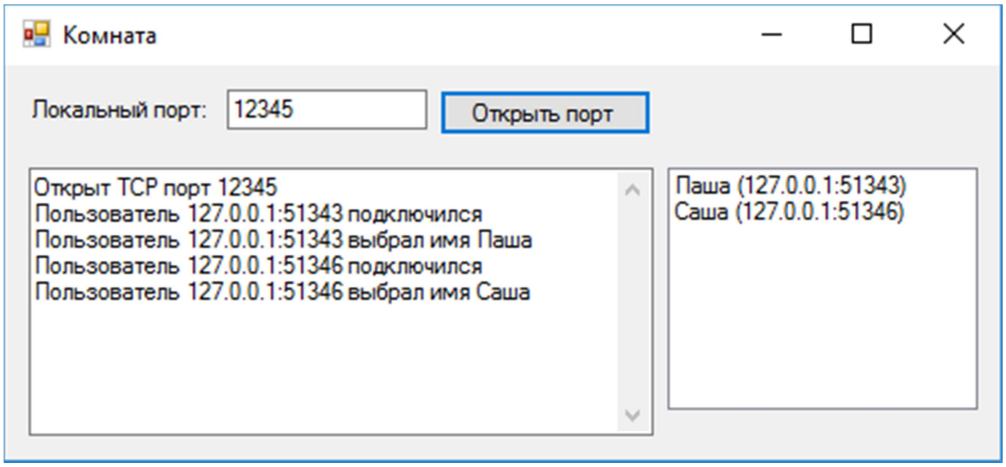
1. Сделайте активным проект для сервера и запустите его в режиме без отладки. Откройте порт 12345.

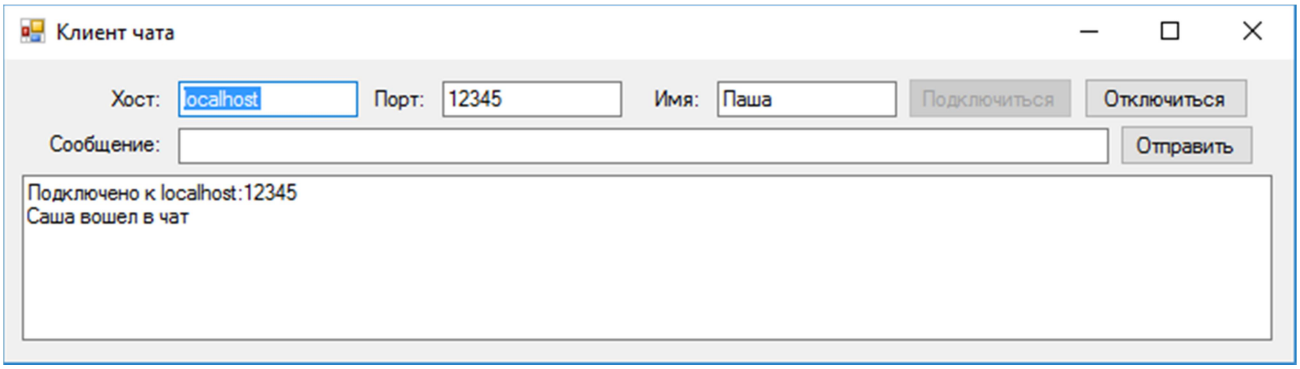


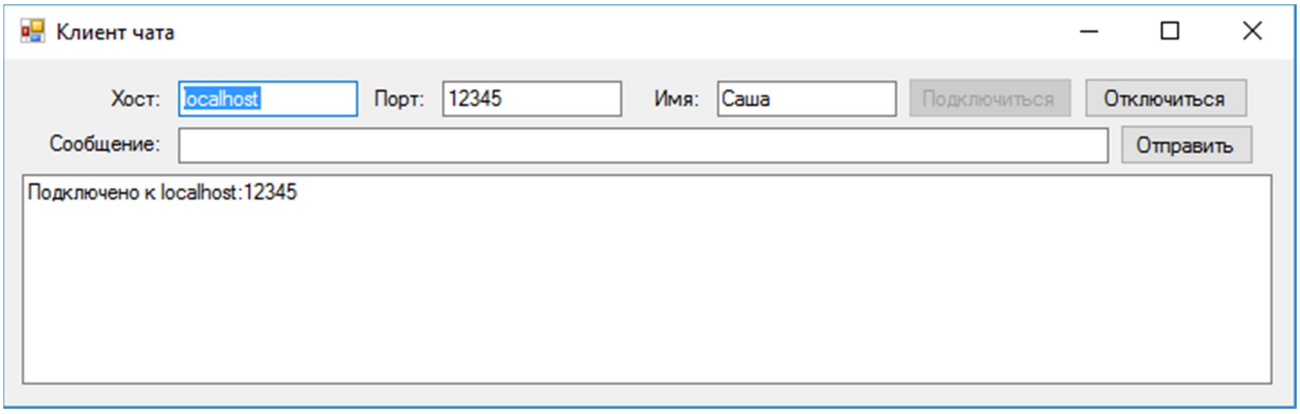
2. Сделайте активным проект клиента и запустите два его экземпляра без отладки.



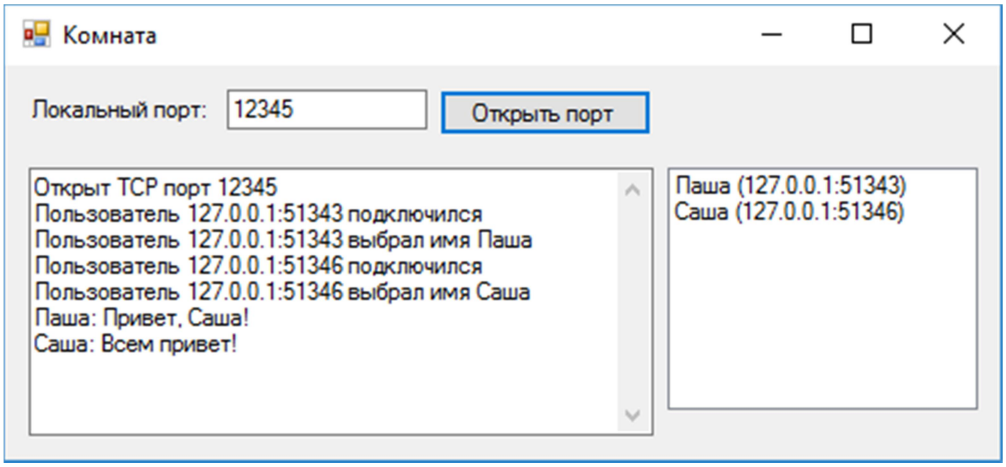
3. Последовательно подключите оба клиента к серверу. После этого приложения должны выглядеть так

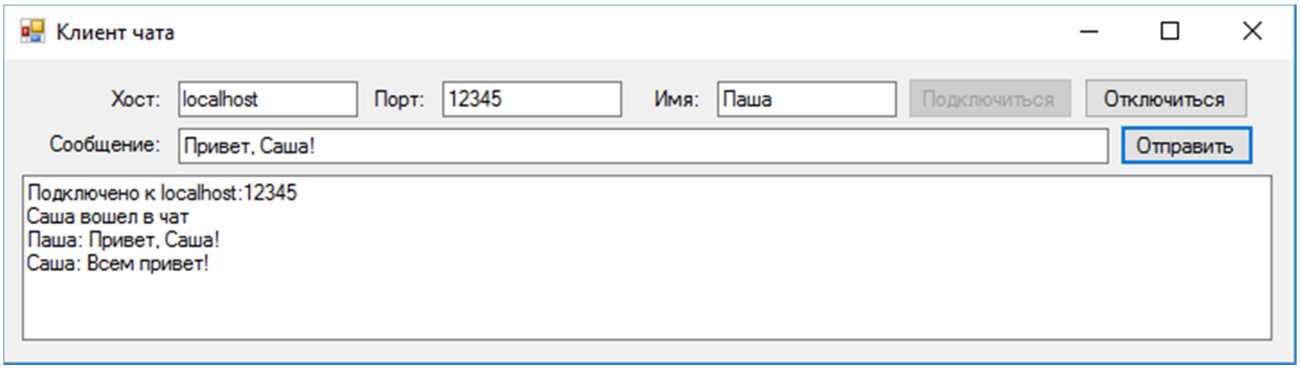


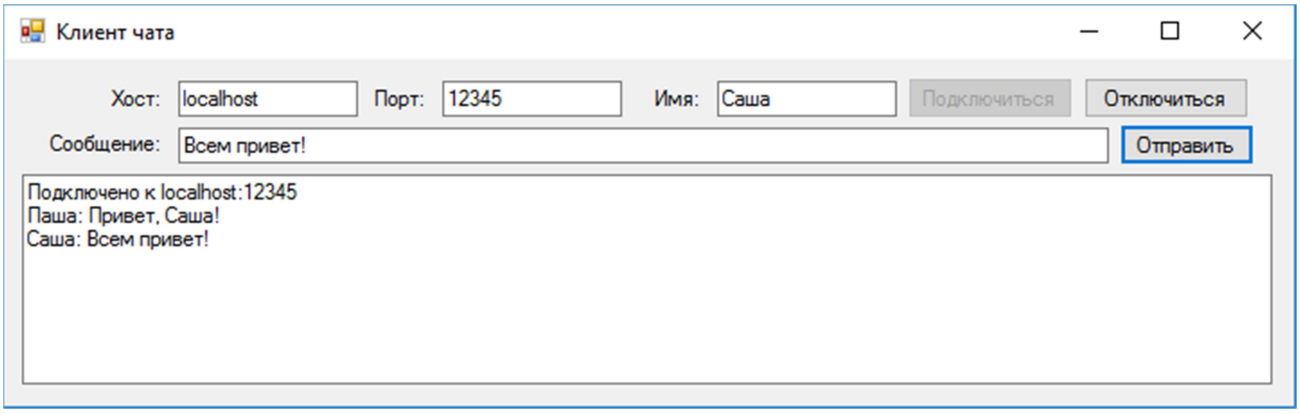




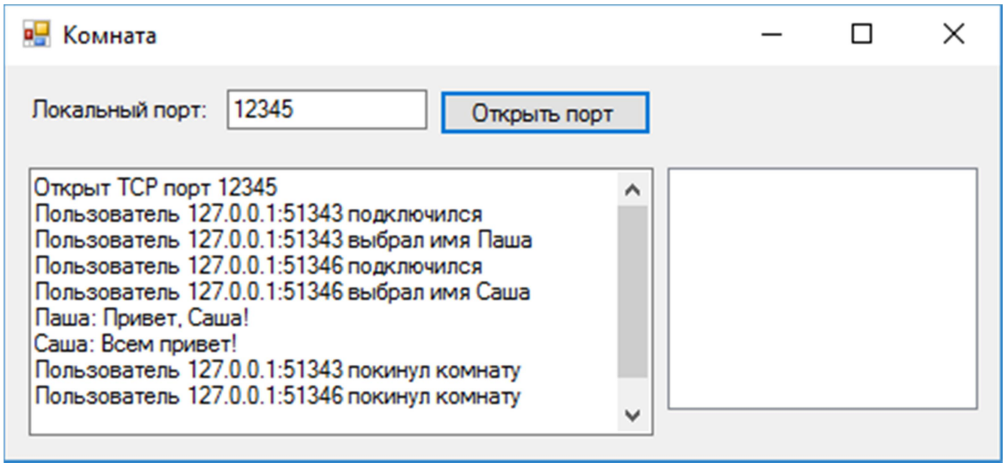
4. Попробуйте отправку сообщений.

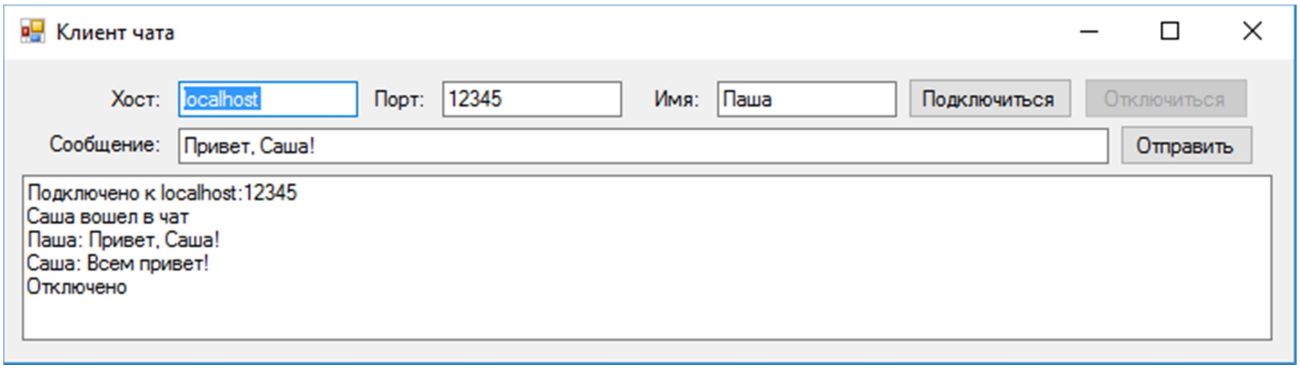


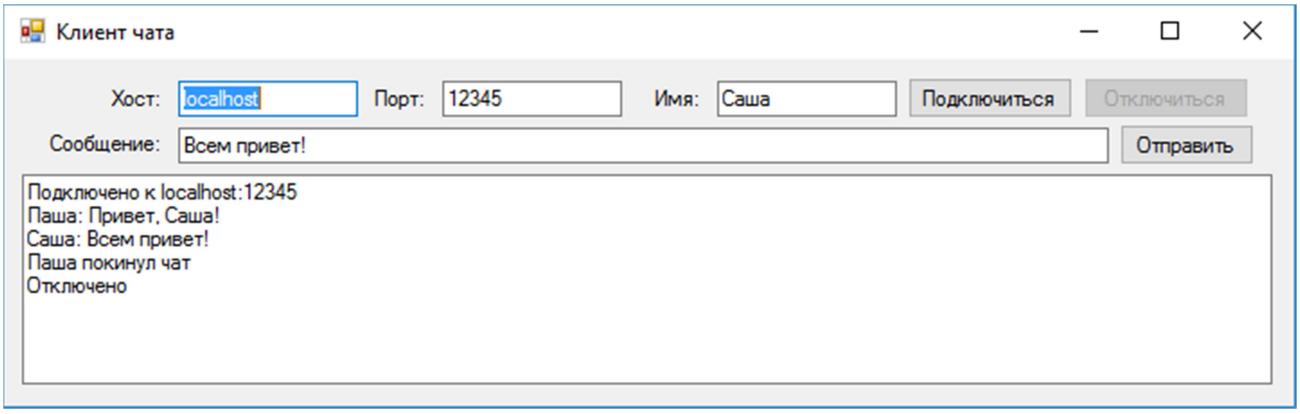




5. Для обоих клиентов разорвите соединение.







6. Попробуйте зайти на сервера на других компьютерах. Пусть кто-то еще зайдет на ваш сервер.