

1 Задание 1. Клиент-серверный чат на основе сокетов

| | | |
|------------|--|-----------|
| 1 | Задание 1. Клиент-серверный чат на основе сокетов | 1 |
| 1.1 | Критерии оценивания..... | 1 |
| 1.2 | Методические указания | 2 |
| 1.3 | Примеры реализации | 2 |
| 1.3.1 | Пример реализации на языке Java..... | 3 |
| 1.3.2 | Пример реализации на языке Python..... | 7 |
| 1.3.3 | Пример реализации на языке C# | 10 |
| 1.4 | Задание на самостоятельную работу | 13 |
| 1.5 | Ссылки | 13 |

Цель: на языке высокого уровня (Java, C#, Python и др. – на выбор обучающегося) реализовать сетевое клиент-серверное приложение – чат (в виде консольного либо диалогового приложения) на основе технологии сокетов. Клиентская программа предоставляет пользователю интерфейс ввода имени пользователя и сообщений, отправляет сообщения серверу и получает от сервера сообщения от остальных пользователей. Соответственно, серверная программа принимает от пользователей сообщения и рассылает их по всем участникам чата, с указанием имени пользователя, который отправил сообщение.

1.1 Критерии оценивания

| № | Задача | Баллы |
|----|--|-------|
| 1. | Реализовать базовое клиент-серверное приложение «Hello World» на основе сокетов. | 5 |
| 2. | Реализовать клиентское и серверное приложение для чата с самим собой. <i>Клиент обеспечивает:</i> 1) Установку имени пользователя; 2) Подключение к серверу по сетевому имени/адресу; 3) Отправку текстовых сообщений от имени пользователя; 4) Получение и отображение сообщений от сервера; 5) Отключение от сервера. <i>Сервер обеспечивает:</i> 1) Подключение одного пользователя; 2) Получение сообщений от пользователя и отправку их обратно; 3) Отключение пользователя от сервера. | 5 |
| 3. | Модифицировать сервер таким образом, чтобы при соединении и разрыве соединения в окно чата всем пользователям отображалось время, логин и сообщение о том что такой-то пользователь «вошел/вышел» | 2 |
| 4. | Сдача в срок: задачи 1-3 сданы до 01 марта – 2 балла; до 15 марта – 1 балл; после 15 марта – 0 баллов. | 2 |

| | | |
|----|--|----|
| 5. | Модифицировать сервер таким образом, чтобы он обеспечивал одновременной общению 2-х и более пользователей одновременно в одной общей комнате. Клиент и Сервер должны поддерживать вывод списка подключенных пользователей. | 3 |
| 6. | Модифицировать клиент и сервер таким образом, чтобы они обеспечивали возможность работы с «Комнатами». Пользователи должны иметь возможность создавать именованные комнаты, изолированные от остальных. Сообщения должны направляться в конкретную комнату чата, и пересылаться только пользователям, подключенным к конкретной комнате. | 3 |
| 7* | Реализовать возможность обмена файлами в рамках комнаты. Пользователь должен иметь возможность загрузить файл на сервер в комнату. В этом случае, все участники чата, подключенные к данной комнате должны иметь возможность загрузить данный файл. Через определенный промежуток времени файл должен удаляться с сервера. | 4* |
| 8* | Реализовать возможность децентрализованного обмена файлами. Файл не загружается на сервер, но формируется специального рода ссылка, обеспечивающая возможность любому участнику чата напрямую загрузить файл с клиента-отправителя, пока клиент подключен к чату и пока на нем доступен данный файл. | 4* |

1.2 Методические указания

Сокет - конечная точка связи двустороннего канала между 2 процессами, выполняющимися либо на одном, либо на разных компьютерах, соединенных сетью. При соединении 2-х сокетов образуется канал, через который можно передавать данные в обе стороны. Одна сторона канала называется **сервером**, другая - **клиентом**. Существует 2 вида сокетов: *потокосые и дейтаграммные*.

Потокосые сокетс работают с установкой соединения, обеспечивая надежную идентификацию обеих сторон, гарантируют целостность и успешность доставки данных. Основываются на протоколе TCP.

Дейтаграммные сокетс работают без установки соединения и не обеспечивают ни идентификации отправителя, ни контроля успешности доставки данных. Ввиду этого они заметно быстрее потокосых. Основываются на протоколе UDP.

Сокетс связываются между собой через порты.

1.3 Примеры реализации

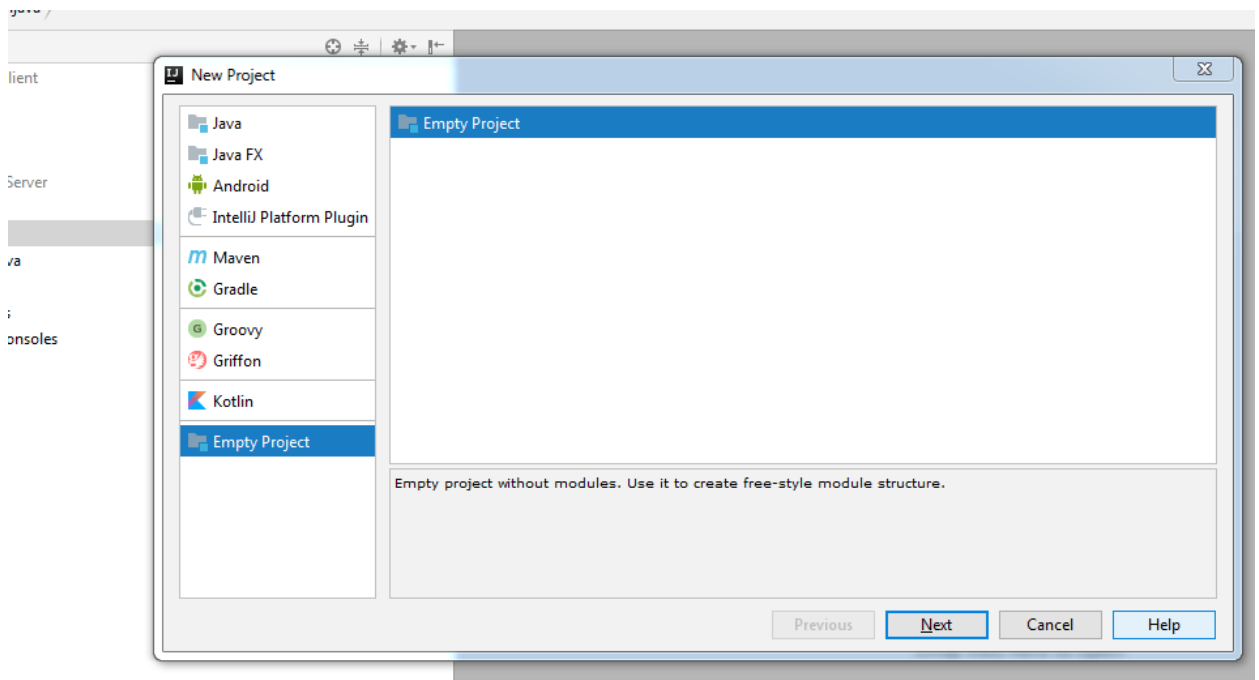
Реализация простого клиент-серверного приложения. Клиент отправляет сообщение серверу и выводит ответ от сервера. Сервер выводит сообщение, полученное от клиента, и отправляет его обратно.

1.3.1 Пример реализации на языке Java

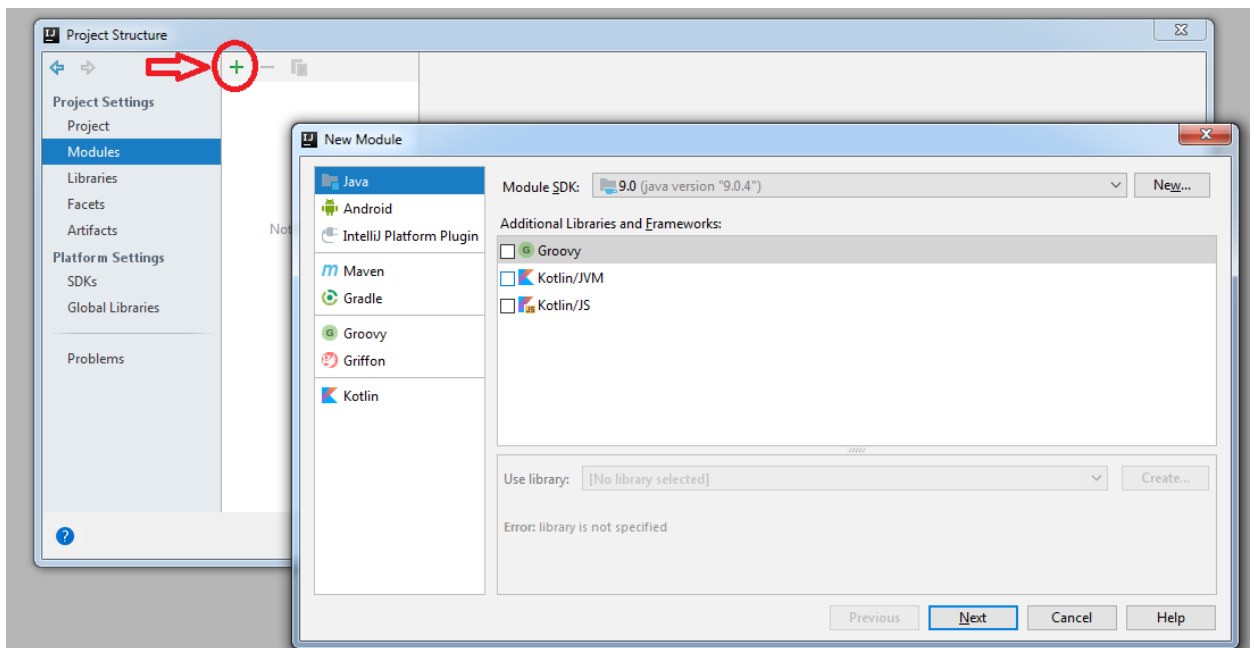
Необходимо загрузить и установить последнюю [Java SDK](#).
IntelliJ IDEA Community Edition вы можете загрузить по [ссылке](#).

Пример реализации клиент-серверного приложения с использованием технологии сокетов на языке Java:

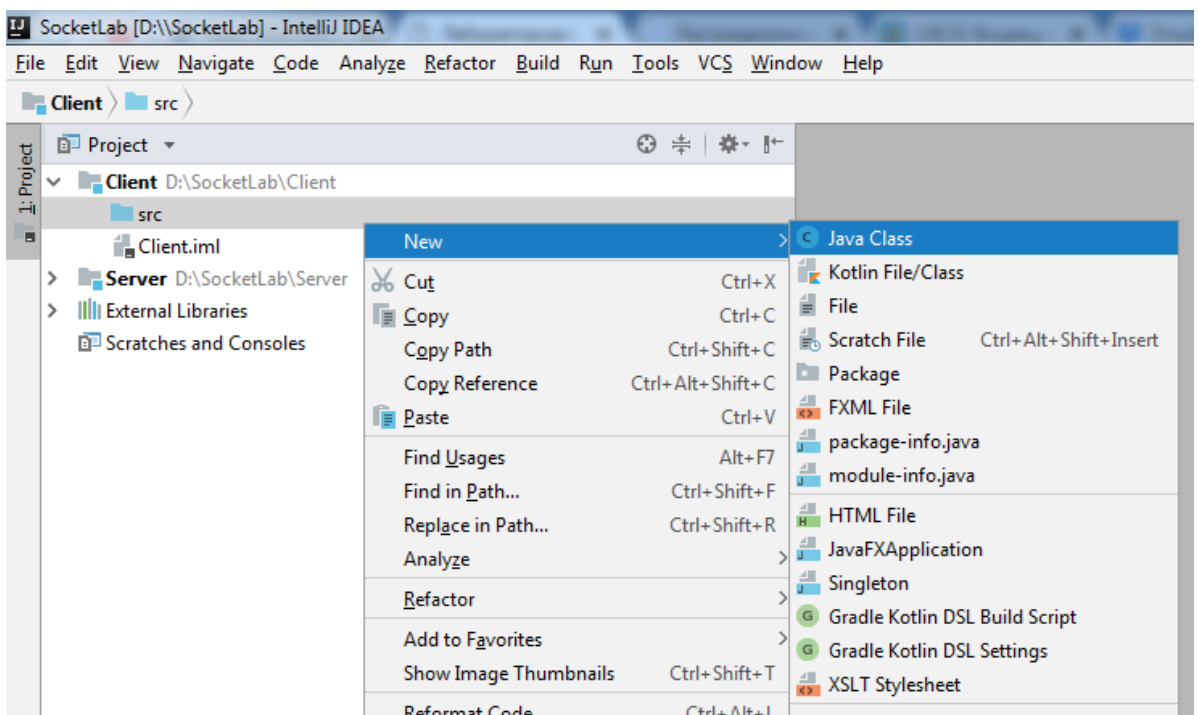
- 1) В среде IntelliJ IDEA создайте новый проект типа «Empty Project» с названием «SocketLab».



- 2) После создания проекта откроется окно «Структура проекта». Добавьте два новых модуля с названиями «Server» и «Client».



- 3) В директории «src» модуля “Server” создайте .java файл с именем “Server”. Аналогично для модуля “Client” с именем “Client”.



- 4) Реализуем исходный код сервера:

- а. Подключим библиотеки для работы с сокетами:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

```
import java.net.ServerSocket;
import java.net.Socket;
```

- b. Укажем порт, подключение на который будет ожидать сервер. Для сервера хост указывать не нужно, он по умолчанию поднимается на “localhost”:

```
public static final int PORT = 19000;
```

- c. В функции main запустим сервер и будем ожидать подключение клиента:

```
public static void main(String[] args) {
    ServerSocket serverSocket = null;

    try {
        serverSocket = new ServerSocket(PORT);

        Socket socket = serverSocket.accept();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- d. После подключения клиента читаем отправленные им данные и выводим их на экран:

```
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();

byte[] buf = new byte[32*1024];
int readBytes = in.read(buf);
String line = new String(buf, 0, readBytes);
System.out.printf("Client> %s", line);
```

- e. После чего отправляем сообщение обратно клиенту:

```
out.write(line.getBytes());
out.flush();
```

В итоге простейший сокет-сервер будет выглядеть следующим образом:

```
public class Server {

    public static final int PORT = 19000;

    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(PORT);

            System.out.println("Started, waiting for connection");

            Socket socket = serverSocket.accept();

            System.out.println("Accepted. " + socket.getInetAddress());

            InputStream in = socket.getInputStream();
```

```

        OutputStream out = socket.getOutputStream();

        byte[] buf = new byte[32*1024];
        int readBytes = in.read(buf);
        String line = new String(buf, 0, readBytes);
        System.out.printf("Client> %s", line);

        out.write(line.getBytes());
        out.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

5) Реализуем исходный код клиента:

а. Исходный код клиента будет похож на исходный код сервера, за исключением явного объявления хоста для сокета:

```

public static final int PORT = 19000;
public static final String HOST = "localhost";

```

и блока отправки сообщения:

```

try (InputStream in = socket.getInputStream();
     OutputStream out = socket.getOutputStream()) {

    String line = "Hello!";
    out.write(line.getBytes());
    out.flush();
}

```

В итоге простейший сокет-клиент будет выглядеть следующим образом:

```

public class Client {

    public static final int PORT = 19000;
    public static final String HOST = "localhost";

    public static void main(String[] args) {
        Socket socket = null;
        try {
            socket = new Socket(HOST, PORT);

            try (InputStream in = socket.getInputStream();
                 OutputStream out = socket.getOutputStream()) {

                String line = "Hello!";
                out.write(line.getBytes());
                out.flush();

                byte[] buf = new byte[32*1024];
                int readBytes = in.read(buf);

                System.out.printf("Server> %s", new String(buf, 0,
readBytes));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

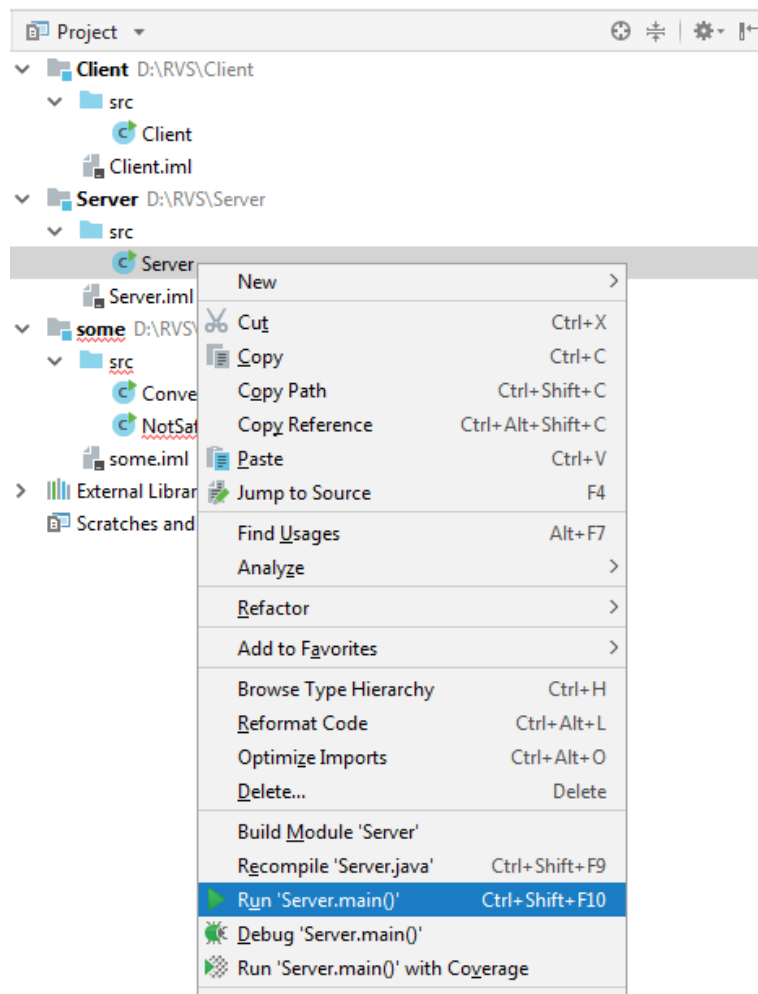
```

```

    }
} catch (IOException e) {
    e.printStackTrace();
}
}

```

- 6) После того, как исходный код клиента и сервера подготовлен, можно запустить приложение. Для этого можно воспользоваться интерфейсом среды IntelliJ IDEA. Нам необходимо запустить в начале сервер, после чего запустить приложение-клиент. Это можно сделать, нажав правой кнопкой на класс «Server» и выбрав: «Run 'Server.main()'» или сочетанием клавиш Ctrl + Shift + F10



После нажатия, запустится сервер и перейдет в режим ожидания соединения.

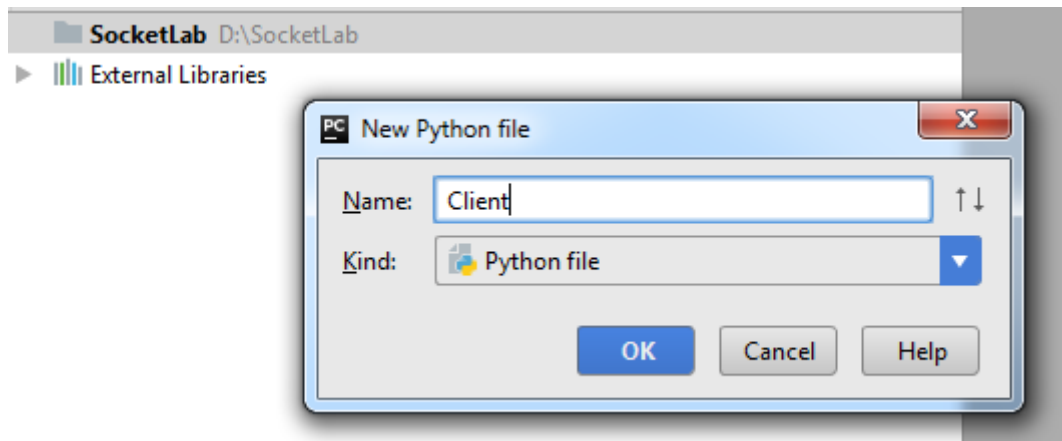
После этого, аналогичным образом, необходимо запустить клиентскую часть приложения.

1.3.2 Пример реализации на языке Python

PyCharm Community Edition вы можете загрузить по [ссылке](#).

Пример реализации клиент-серверного приложения с использованием технологии сокетов на языке Python:

- 1) В среде PyCharm создайте новый проект с названием “SocketLab”.
- 2) После создания проекта добавьте два Python файла с названиями “Server” и “Client”.



- 3) Реализуем исходный код сервера:

- a. Подключим библиотеку для работы с сокетами:

```
import socket
```

- b. Создадим сокет:

```
sock = socket.socket()
```

- c. Теперь свяжем наш сокет с хостом и портом с помощью метода bind, которому передается кортеж, первый элемент которого — хост, а второй — порт:

```
sock.bind( ('', 9090) )
```

Насчет хоста — мы оставим строку пустой, чтобы наш сервер был доступен для всех интерфейсов.

- d. С помощью метода listen мы запустим для данного сокета режим прослушивания. Метод принимает один аргумент — максимальное количество подключений в очереди:


```
sock.listen(1)
```

- е. Теперь мы можем принять подключение с помощью метода `accept`, который возвращает кортеж с двумя элементами: новый сокет и адрес клиента:

```
conn, addr = sock.accept()
```

- ф. Для чтения данных используется функция `recv`, которой первым параметром нужно передать количество получаемых байт данных:

```
data = conn.recv(1024)
```

Тип возвращаемых данных — `bytes`. У этого типа есть почти все методы, что и у строк, но для того, чтобы использовать из него текстовые данные с другими строками, придётся декодировать данные и использовать уже полученную строку

```
uData = data.decode("utf-8")  
print("Client > " + uData)
```

- г. Для отправки данных в сокет используется функция `send`. Принимает она тоже `bytes`, поэтому для отправки строки вам придётся её закодировать:

```
conn.send(uData.encode("utf-8"))
```

- х. После всего и клиенту, и серверу необходимо закрыть сокет с помощью функции `close`:

```
conn.close()
```

В итоге простейший сокет-сервер будет выглядеть следующим образом:

```
import socket  
  
sock = socket.socket()  
sock.bind(('', 9090))  
  
sock.listen(1)  
  
print('waiting for connection...')  
conn, addr = sock.accept()  
  
print('connected: ', addr)  
  
data = conn.recv(1024)  
uData = data.decode("utf-8")  
print("Client > " + uData)  
conn.send(uData.encode("utf-8"))  
  
conn.close()
```

4) Реализуем исходный код клиента:

а. Исходный код клиента будет похож на исходный код сервера:

```
import socket

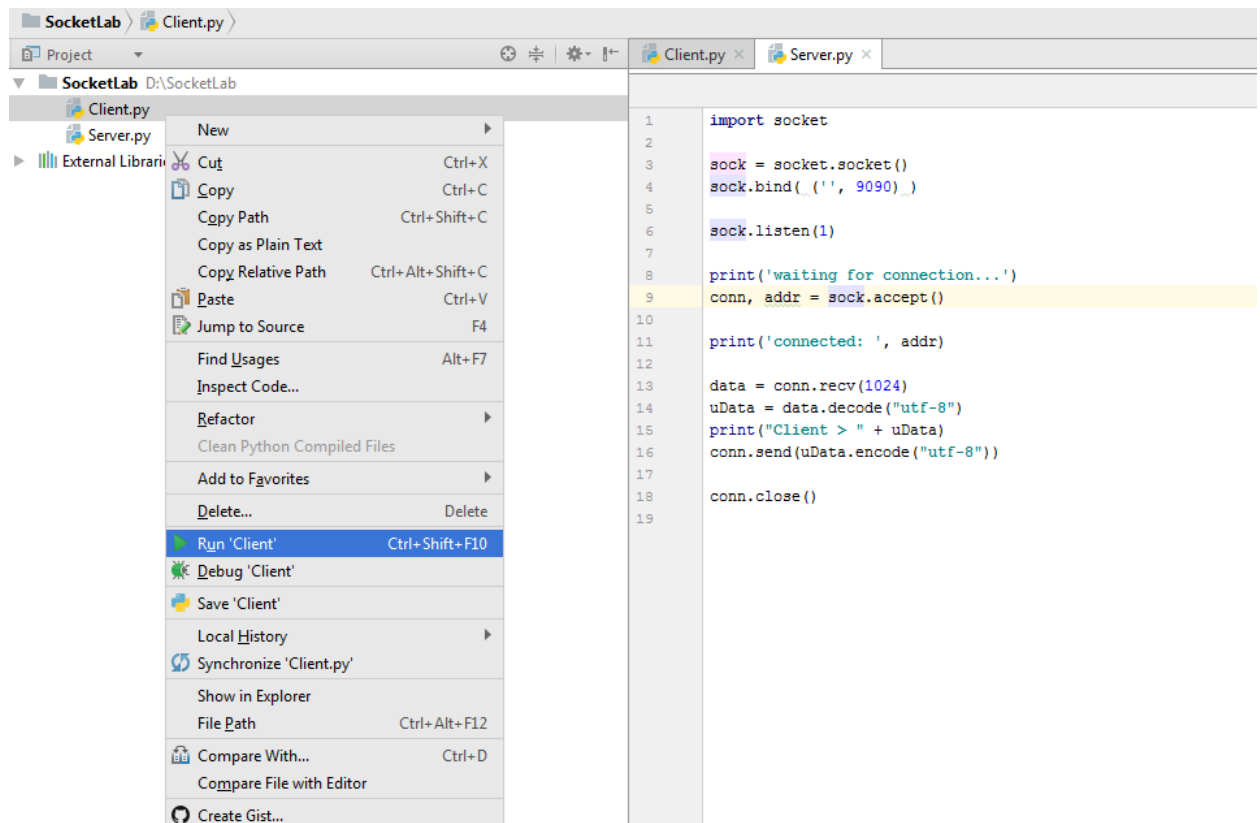
sock = socket.socket()
sock.connect(('localhost', 9090))

sock.send(b'Hello!\n')

data = sock.recv(1024)
udata = data.decode("utf-8")
print("Server > " + udata)

sock.close()
```

5) После того, как исходный код клиента и сервера подготовлен, можно запустить приложение. Для этого необходимо запустить в начале сервер, после чего запустить клиент. Это можно сделать, нажав правой кнопкой на файл «Server.py» и выбрав: «Run 'Server'» или сочетанием клавиш Ctrl + Shift + F10



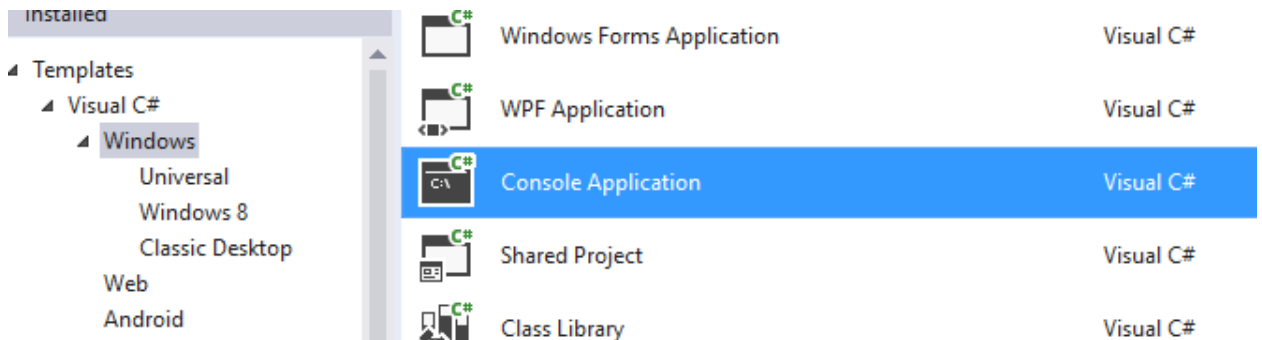
После нажатия, запустится сервер и перейдет в режим ожидания соединения.

После этого, аналогичным образом, необходимо запустить клиентскую часть приложения.

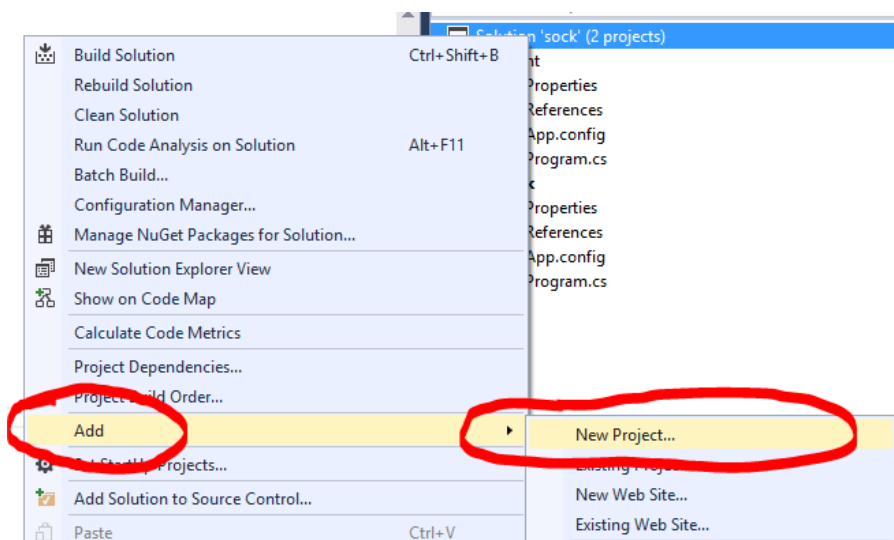
1.3.3 Пример реализации на языке C#

Пример реализации клиент-серверного приложения с использованием технологии сокетов на языке C#:

- 1) В среде Visual Studio создайте новый проект типа «Консольное приложение» под названием «SocketLab».



- 2) В созданном решении будет создан один проект по умолчанию. Для клиент-серверного приложения нам необходимо создать 2 независимых проекта (один – для клиента, другой – для сервера). Таким образом, добавим еще один проект в наше решение, нажав правой кнопкой по решению, и выбрав «Добавить...-> Новый проект»



В открывшемся окне, создадим еще одно консольное приложение, назвав его «SocketClient».

- 3) Реализуем исходный код сервера:

- а. В блок using необходимо добавить библиотеки для работы с сокетами:

```
using System.Net;  
using System.Net.Sockets;
```

b. В функцию main добавим информацию о конечной точке нашего сервера:

```
Int32 serverPort = 13000;  
IPAddress localAddr = IPAddress.Parse("127.0.0.1");  
TcpListener calcServer = new TcpListener(localAddr, serverPort);
```

c. Запустим сервер и будем ожидать подключения клиента:

```
calcServer.Start();  
TcpClient client = calcServer.AcceptTcpClient();  
NetworkStream stream = client.GetStream();
```

d. После подключения клиента, считаем отправленные им данные и выведем их на экран:

```
Byte[] bytes = new Byte[256];  
string data = null;  
int i = stream.Read(bytes, 0, bytes.Length);  
data = System.Text.Encoding.UTF8.GetString(bytes, 0, i);  
  
Console.Write(data);
```

В итоге, основной метод простейшего сокет-сервера может выглядеть следующим образом:

```
Int32 serverPort = 13000;  
IPAddress localAddr = IPAddress.Parse("127.0.0.1");  
TcpListener calcServer = new TcpListener(localAddr, serverPort);  
  
calcServer.Start();  
TcpClient client = calcServer.AcceptTcpClient();  
NetworkStream stream = client.GetStream();  
  
Byte[] bytes = new Byte[256];  
string data = null;  
int i = stream.Read(bytes, 0, bytes.Length);  
data = System.Text.Encoding.UTF8.GetString(bytes, 0, i);  
  
Console.Write(data);  
Console.ReadKey();
```

4) Реализуем исходный код клиента:

a. Исходный код клиента будет напоминать исходный код сервера, за исключением блока подключения к серверу:

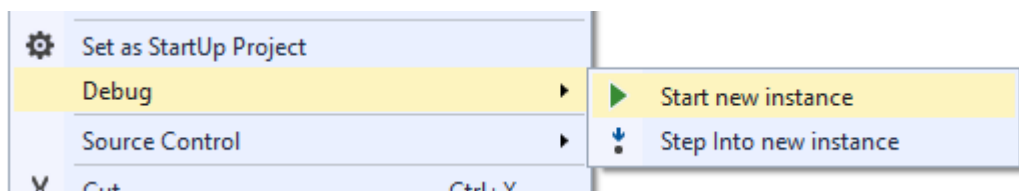
```
...  
TcpClient client = new TcpClient();  
client.Connect(serverAddr, serverPort);  
NetworkStream stream = client.GetStream();  
...
```

и блока отправки сообщения:

```
...  
String data = "Socket Hello World";  
bytes = System.Text.Encoding.UTF8.GetBytes(data);  
stream.Write(bytes, 0, bytes.Length);  
  
client.Close();  
...
```

По данным указаниям, вам предлагается самостоятельно реализовать простейший сокет-клиент, отправляющий одиночное сообщение на сервер.

- 5) После того, как исходный код клиента и сервера подготовлен и собран, можно запустить приложение. Для этого можно воспользоваться интерфейсом среды Visual Studio. Нам необходимо запустить в начале сервер, после чего запустить приложение-клиент. Это можно сделать, нажав правой кнопкой на проект «SocketLab» и выбрав: «Отладка->Запуск нового экземпляра»



После нажатия, запустится сервер и перейдет в режим ожидания соединения.

После этого, аналогичным образом, необходимо запустить клиентскую часть приложения, нажав правой кнопкой по проекту «SocketClient» и выбрав аналогичный пункт меню. Альтернативно, клиент и сервер можно запустить непосредственно из папки проекта, запустив исполняемые файлы SocketLab.exe и SocketClient.exe из директорий bin\Debug соответствующих проектов.

1.4 Задание на самостоятельную работу

На основе разработанного простейшего приложения с использованием технологии сокетов, вам необходимо реализовать сетевое клиент-серверное приложение – чат (в виде консольного либо диалогового приложения) на основе технологии сокетов.

1.5 Ссылки

[Пространство имен System.Net.Sockets \(C#\)](#)

[Сокеты в Python](#)