

TP1: Socket Programming in Java

The objective is to learn how to create a simple Server and clients that connects to each other with Sockets over TCP using Java. To use Java Programming language, you need to install the Java Development Kit (JDK) as well as a programming editor (IDE) such as Eclipse.

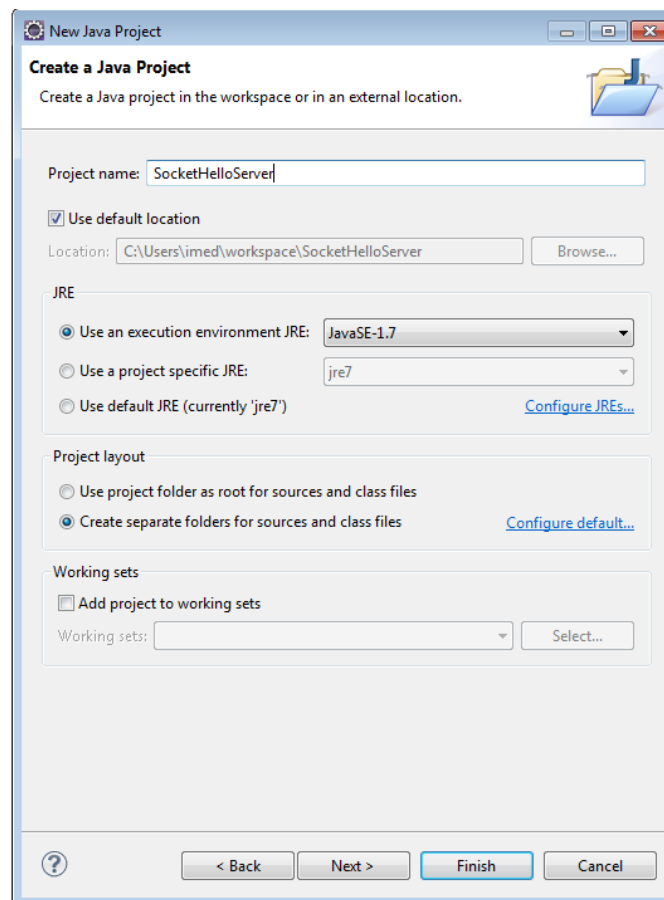
Part I: Simple server using TCP sockets

The objective of this part is to create a simple server that can add two given numbers following these steps:

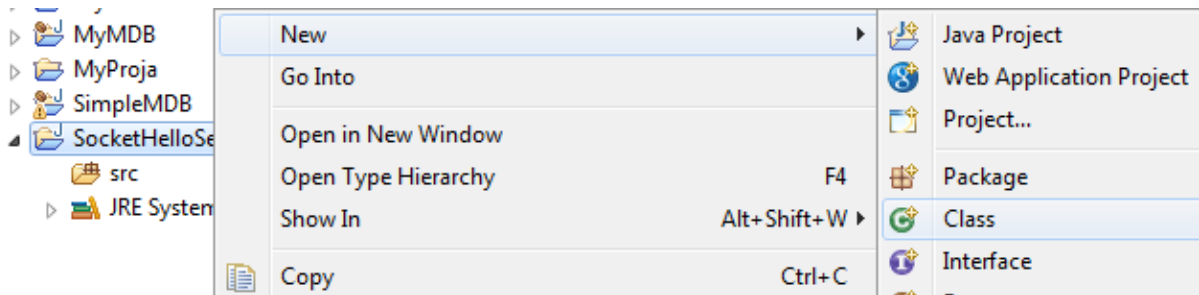
- Creating and Running a Server
- Connecting to the server using PuTTY
- Creating a Java Client.

I.1. Creating and Running a Server

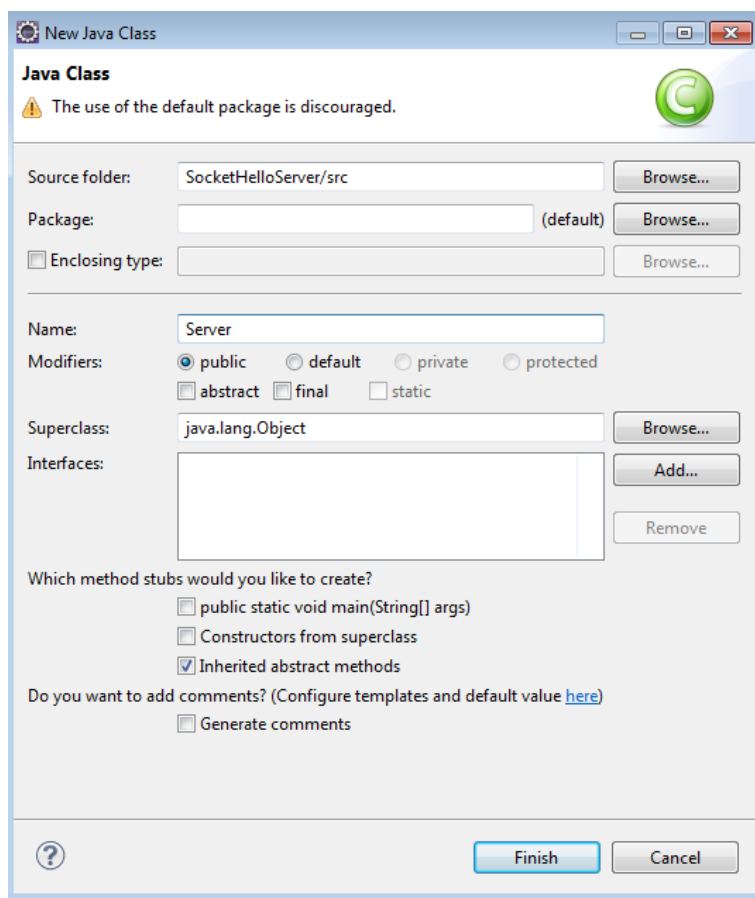
1) Create a new project using Eclipse (or NetBeans or other editor you prefer), and call it : **SocketHelloServer** -> Click **Finish** once done



2) Within the new project: **Right Click** on the project name within the **Package Explorer**, Choose, **New** then **Class**



3) Type in the name of the class as : **Server** Then click **Finish**



4) Write the code for the Server that does the Addition operation. Copy the following code into the **Server** class that you had just created.

```
1. import java.io.*;
2. import java.net.*;
3.
4. class Server
5. {
6.     public static void main(String argv[]) throws Exception
7.     {
8.
9.         System.out.println(" Server is Running " );
10.        ServerSocket mysocket = new ServerSocket(5555);
```

```

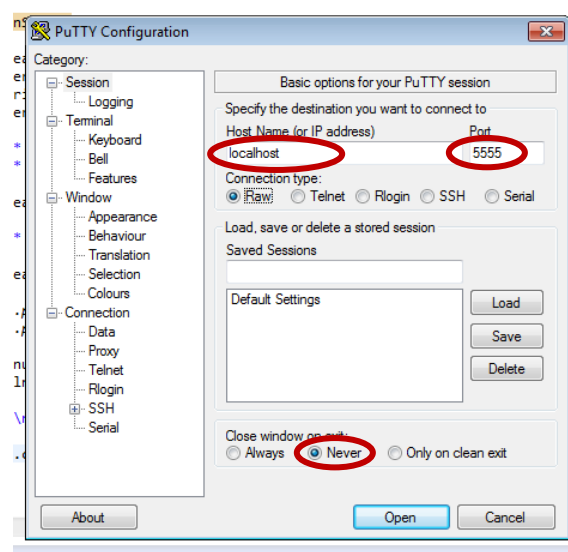
11.
12.         while(true)
13.         {
14.             Socket connectionSocket = mysocket.accept();
15.
16.             BufferedReader reader =
17.                 new BufferedReader(new
18. InputStreamReader(connectionSocket.getInputStream()));
19.             BufferedWriter writer=
20.                 new BufferedWriter(new
21. OutputStreamWriter(connectionSocket.getOutputStream()));
22.
23.             writer.write("*** Welcome to the Calculation Server (Addition Only)
24. ***\r\n");
25.
26.             writer.write("*** Please type in the first number and press Enter : \n");
27.             writer.flush();
28.             String data1 = reader.readLine().trim();
29.
30.             writer.write("*** Please type in the second number and press Enter :
31. \n");
32.             writer.flush();
33.             String data2 = reader.readLine().trim();
34.
35.             int num1=Integer.parseInt(data1);
36.             int num2=Integer.parseInt(data2);
37.
38.             int result=num1+num2;
39.             System.out.println("Addition operation done " );
40.
41.             writer.write("\r\n=== Result is : "+result);
42.             writer.flush();
43.             connectionSocket.close();
44.         }
45.     }
46. }

```

5) Compile and Run the Application now.

I.2. Connecting to the server using PuTTY

PuTTY is a simple terminal program that we use to connect to other machines usually through SSH, Telnet and so on. (1) Download PuTTY. (2) Double Click on PuTTY to start it. Fill it as shown below. The address is usually **localhost** for your computer. Choose **RAW** for the connection type. The port is **5555** as shown in the code above. Choose **NEVER** for the closing of the window.

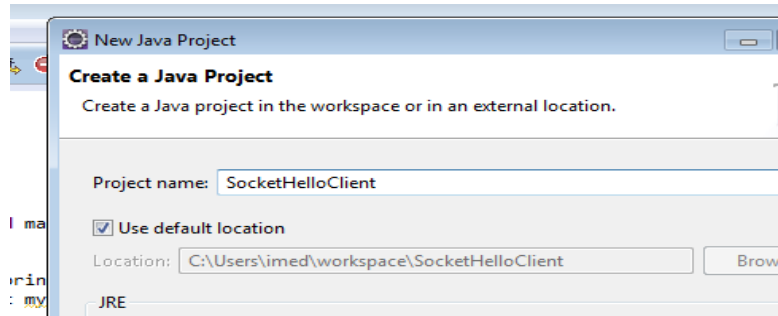


3) Once you have filled all the required information, click on Open. A black window should shown up asking for the first number...

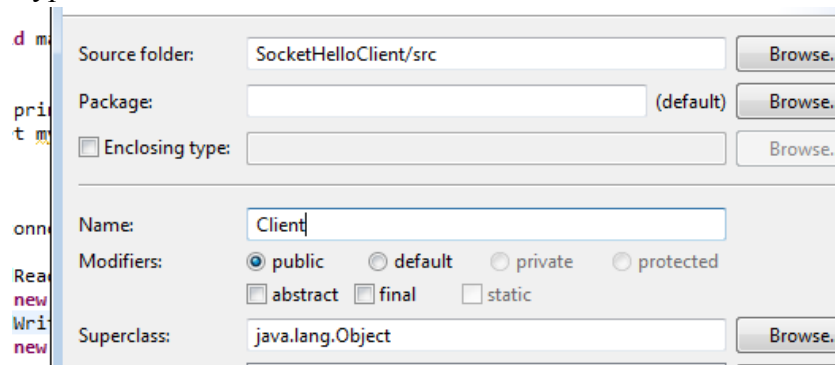
I.3. Creating a Java Client

Instead of using PuTTY as a client for the Addition Server, we can write **Java Client** through the following steps.

1) Within Eclipse, Create NEW Java Project under the name : SocketHelloClient



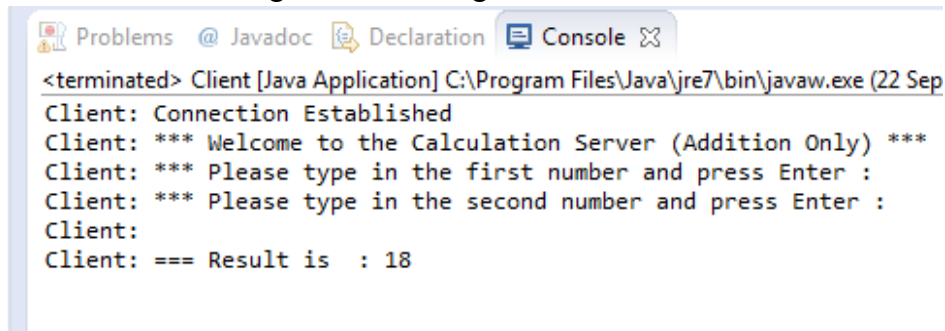
2) Right Click on the project name **SocketHelloClient** inside the Package Explorer, choose **New -> Class**. Type the name of the class as **Client**



3) Copy the following code the Client class. The Client is programmed to send two numbers: **8** and **10** to the server for the Addition operation.

```
1. import java.io.*;
2. import java.net.*;
3. import java.util.*;
4.
5. public class Client {
6.
7.     public static void main(String argv[])
8.     {
9.         try{
10.             Socket socketClient= new Socket("localhost",5555);
11.             System.out.println("Client: "+"Connection Established");
12.
13.             BufferedReader reader = new BufferedReader(new
InputStreamReader(socketClient.getInputStream()));
14.
15.             BufferedWriter writer= new BufferedWriter(new
OutputStreamWriter(socketClient.getOutputStream()));
16.             String serverMsg;
17.             writer.write("8\r\n");
18.             writer.write("10\r\n");
19.             writer.flush();
```

- 4) Make sure that the server is running!
- 5) Run the client now. You will get the following into the console:



```
<terminated> Client [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (22 Sep 2015 10:10:10 AM)
Client: Connection Established
Client: *** Welcome to the Calculation Server (Addition Only) ***
Client: *** Please type in the first number and press Enter :
Client: *** Please type in the second number and press Enter :
Client:
Client: === Result is : 18
```

Part II: Threaded Server using TCP Socket

In part I, we have created a simple server using TCP sockets. Because of the limitation of accepting only a single client at a time, we will show in this part how to create a multi-threaded server that can handle multiple client connection at the same time.

- 1) Create a new project named **ThreadedServer**. Then, create a class named: **Server**.
- 2) Copy the code (Server.java Part I) in the Server Class.
- 3) Modify the code in order to support Multi-threading. Multi-threading is a method of writing code for executing tasks in parallel. (For help) Consider the following code segments:

class Server implements Runnable To implement a thread, the class needs to implement the **Runnable** interface **OR** Extends the **Thread** class.

public Server(Socket s){ Creating a constructor for the server class which takes the object Socket as an argument

public void run(){ This is the golden method of any thread. When making a thread, the core code to be executed needs to be placed within the **run** method.

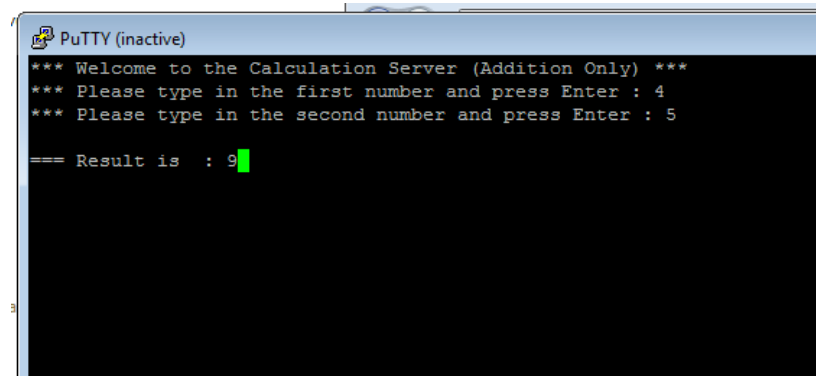
Socket sock = mysocket.accept(); The server is waiting for client connection to be accepted. Once a successful connection is made, the socket object is created.

Thread serverThread=new Thread(server); Because we chose to implement the Runnable interface, we need to create a Thread object. The thread object usually takes as argument any class that implements the Runnable interface.

serverThread.start(); This method (**start()**) would trigger the execution of the code within the **run()** method.

- 3) Run the Server now From Eclipse.

4) Open TWO Putty terminal windows and connect to the server. The address is usually **localhost** for your computer. Choose **RAW** for the connection type. The port is **5555** as shown in the code above. Choose **NEVER** for the closing of the window. Once you have filled all the required information, click on **Open**. A black window should show up asking for the first number and so on...



```
PuTTY (inactive)
*** Welcome to the Calculation Server (Addition Only) ***
*** Please type in the first number and press Enter : 4
*** Please type in the second number and press Enter : 5

=== Result is : 9
```

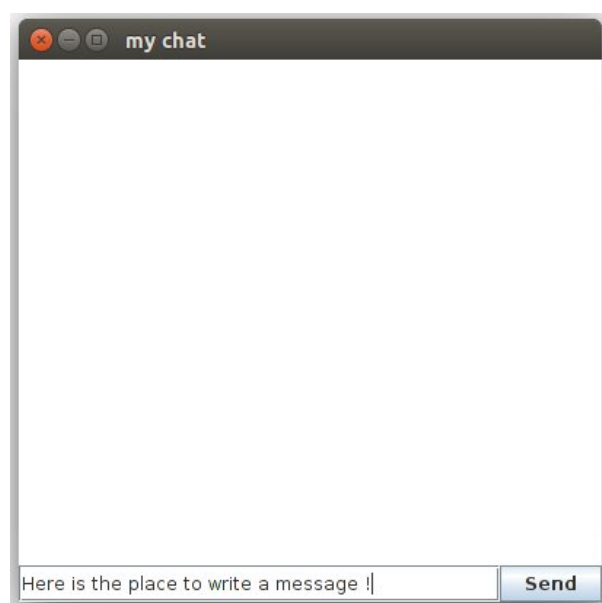
Part III: Group chat system using low-level sockets

In order to create the group chat system using low-level sockets, you will be having three simple steps.

- 1) Creating the user interface
- 2) Creating the Chat Server
- 3) Broadcasting Messages to All Clients

III.1. Creating a Java Client

1) Create a simple graphical interface for the client to send and receive the chat messages ("chat.java"). Then, create an instance of the interface and starts the client. Once you when you run the code, you shall get the following user interface.



You have now to complete “chat.java” file.

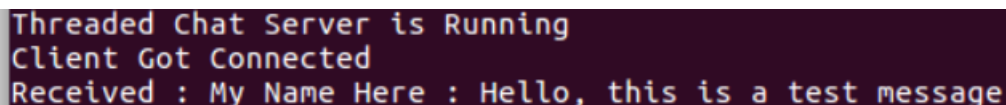
3) Make a simple event handler for sending a message from the client to the server when the user clicks the **send** button after typing some text. You must define the instance variable for a **BufferedWriter**.

4) Create the writer object which writes to the server. Within the constructor, we add the following code. The **localhost** address can be changed to reflect the server address. 5555 is the chosen port number.

5) Add the action listener to the button (**Send**).

III.2. Creating the Chat Server

Create a server to listen to clients sending messages. The server needs to be started the first. This is the screenshot from the server side receiving the message from the client:



```
Threaded Chat Server is Running
Client Got Connected
Received : My Name Here : Hello, this is a test message
```

III.3. Broadcasting Messages to All Clients

The server can receive the messages sent by the client. The server now needs to broadcast every received message back to all clients. Complete the code using these steps:

- 1) Create a vector as a static variable to store all clients.
- 2) Once we get a client connected, we would store its **BufferedWriter** object inside the **Vector** list.
- 3) The server should iterate through all writer objects and sends the message back to the clients.
- 4) On the client side, you need to listen to the server when it sends back message. A better way would be to create a thread running in parallel with the **BufferedReader** instance (chat implements **Runnable**).