CS311 Socket Lab

In this lab you will compile and run some example Java code that sets up a socket connection between two machines in the lab.

Background

Sockets provide a communication mechanism between two computers using TCP/IP (transmission control protocol).

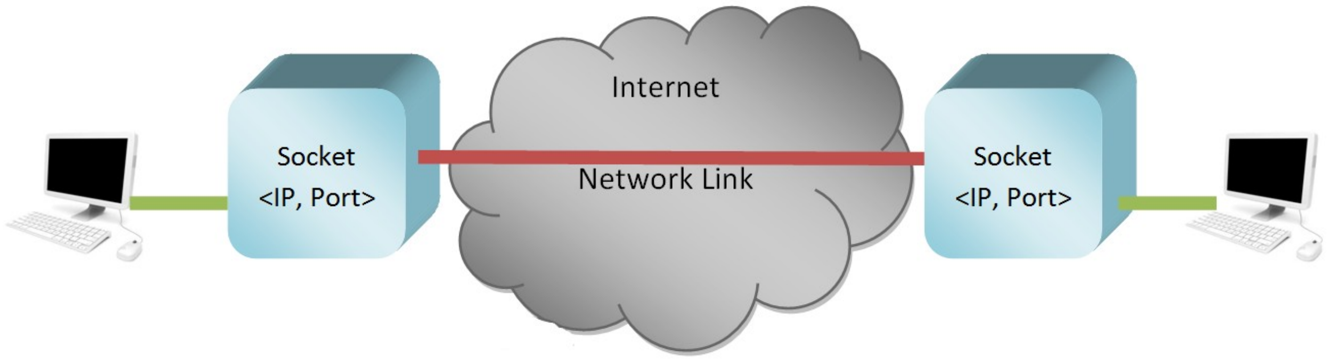
A client program creates a socket on its end of the communication and attempts to connect to a server. When a connection is made, the server creates a socket object on its end of the communication. The client and the server can then communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

To communicate with a server socket on another machine, you have to tell your client code what the server IP address and the TCP port number are. With this info, you can set up the connection from one machine to another.

At the software level, within an operating system, a port is a logical construct that identifies a specific process that is listening for connections on a server. A software port is associated with the IP address of a host. There can be more than one server port listening per IP address.

For example, I can create a socket connection from a client to a server by having the client code connect to a server at IP address 192.168.1.1 on destination port 5004. I could have a different client connect to the same server (192.168.1.1), but on a different destination port 5005.



The server sample code provided with this lab sets up a server socket which starts a thread for each connection attempt that it receives. The server thread creates the server end of the socket, reads a message off that socket and then responds by sending a message back through the socket.

The sample client code connects to the server, sends a string message, and then reads a reply from the server and prints it.

Instructions:

Download the following files: MyServer.java, MyClient.java, ServerThread.java

To start the lab, we'll setup a socket connection between a client and server on the *same* machine.

Part 1:

Open two Linux terminal windows.

In each window, navigate to the directory where you downloaded the files above.

In terminal 1:

Compile the code: **`javac *.java`**

Pick a port number in the range 9001 to 9999 to use as the *myPort* variable on the following line. For example, 9998.

Run the server: **`java MyServer`** *`myPort`*

Again, *myPort* should be a number between 9001 and 9999, NOT the word *myPort*!
If the server starts properly, you should see a message like the following:

```
The server is starting....
```

In terminal 2:

Now that our server is running, we can connect to it using the client code. The client will open a network socket connection to the server and then send the server a message. Once the server reads the message, it will respond back to the client with a message, which the client will print before it exits.

Find the IP address of your machine: use the ***ifconfig*** linux command to do this. You might need to type the full path for ifconfig: /sbin/ifconfig

The IP address is shown in the output from ***ifconfig.*** For example, it is 10.6.20.21 in the example below:

```
eth0      Link encap:Ethernet  HWaddr f8:bc:12:78:e8:c4
          inet addr:10.6.20.21  Bcast:10.6.20.255  Mask:255.255.255.0
```

Run the client: **`java MyClient 10.6.20.21 myPort hellofromclient`**

where *myPort* is the same number that you picked for your server above in the other terminal. and the IP address is the one for your machine.

If the client starts properly, you should see the following in the client terminal window.

```
The client is starting....
sending message to: 10.6.20.21:9998, hellofromsteve
got message back from server: Hello, you are client #1.
closing socket
client done
```

At the same time, in the server window, you should see the following:

```
Server thread starting, serving client: 1
     server read a line: hellofromsteve
     server read a line: .
     server thread done
```

Part 2:

Now that you are able to communicate from client to server via a socket on your own machine, try connecting to one of your class mate's servers. To do this, you'll need to ask them their IP address and port number. (You can get your IP address by using the “ifconfig -a” command).

All you have to do then is to tell your client where to connect to using the command line arguments, like this:

```
java MyClient yourclassmatesIP  yourclassmatesport  takethatyoucriminal
```

Send a message to at least three of your classmates and verify that they received it.

Part 3:

Now that you can communicate to someone else's server socket, try connecting to Mr. Sheehy's server.

You can send the Sheehy server a message to get back your very own custom message.

Note that being a very “proper” server, this server will only respond back to you if the message you send is properly formatted. Otherwise it just sends back a “.”.

In order to get a response from Mr. Sheehy's server, you must first send it a message to get the proper protocol. To do this, send a message like the following:

```
yourfirstname|PROTOCOL
```

If properly formatted, you'll get a message back telling you how to format another message you can send to the Sheehy server to get back your own custom code.

Once you have determined your own custom code, fill your code in on the table on the chalk board.

Part 4:

Try spamming someone else's server (get permission from them first!). See if you can crash your classmate's server or make it unresponsive. You'll have to modify your client code significantly to do this....think “sending many messages in a loop”.

Note that if someone else is attacking you, that you might be able to defend against this attack by having your server spawn more threads. Again, this will require some modification to your server code.

Part 5:

As an extra challenge, rewrite your client code in another language. For example, try using C or C++ or Python to write your client. If you get this part working let me know.

Useful commands to try:

```
/sbin/ifconfig -a
netstat -i
ping  IP_address (eg: ping -vsn 192.168.1.1)
```