Simple Clients and Servers in Java

Objectives:

1. Design and implement a simple server in Java.

2. Design and implement a simple client in Java.

Preparation: Before doing this lab read the following pages in *Java: How to Program* by Deitel and Deitel, sixth edition: pages 1106-1108 for introduction on networking in Java; pages 1108-1132 on clients and servers, pages 1052-1062 on Java threads.

Bring the Java text to lab.

Laboratory Exercices:

In this lab you will construct a Java application for a simple server and a Java application for a simple client which communicates with the server. **NO graphics in the exercise!** Graphics were left out on purpose!

1. The Client Application: Write a Java application which implements a simple client. The client should connect to your server that will be running on another host, e.g., **linuxcomp3**. Use any **port** between 8000 and 9999. [1] Note that only one server on a host may use a specific port number. Therefore, if you receive a message of "port in use", just select another port from 8000 to 9999.

In the client, hard code which host will be running your server.

The client should repeatedly read a line a text from **System.in** and send the line to the server. Then the client receives a new line of text from the server and prints it to **System.out**. If the user types exactly "quit", the client sends it to the server to close its connection and the client should close its connection and exit.

Using the example of a client on pages 1117-1132 as a model, write the Java code and compile it. If the server is not running, when you run the client, you should have it print out a message on **System.err** something like "Server is down. Make sure server is running first!".

2. The Server Application: Write a Java application which implements a simple server. Once the socket has been opened, the server waits for the client. After a connection with the client, the server repeatedly receives a line of text from the client, changes the text in some way, and sends it back to the client until the client sends exactly "quit". When the server receives "quit", it sends a usual message to the client, closes the client's connection and waits for another client.

The server should print messages to **System.out** to show what it is doing. For example: "Waiting for a client ..." "Connected with client on lemon".

Using the example of a server on pages 1117-32 as a model, write the Java code and compile it. Run the server first on the proper host, e. g., **linuxcomp3**, then run your client on a different host.

Note: Your server application should only be able to handle one client at a time. After one client disconnects, the server should be be able to handle a second client and so on. Use **Control-c** to kill your server when needed.

[1] Normally, you could use any port from 1024 to 65535 but ISR has a fi rewall blocking many of the ports.

3. Using Someone Else's Server: After you have your server and client running properly, ask someone else in class for the host and port of their server and try your client on it. Does it work?

Hand in

Show your lab instructor that your client and server work. If you are unable to finish the lab within the lab time, email me the two files for your client and server with instructions on how to run the two. Since I will test them carefully, you may prefer to show me in the lab. :^)

For your working client and server, hand in the java code. Print the java code using **a2ps**. Handin the output from both the client and the server of a session.

In a sentence or two, describe your experience with trying to connect your client to someone else's server.