CS3004 Network Computing

EchoServer

# Main Objective of the exercise

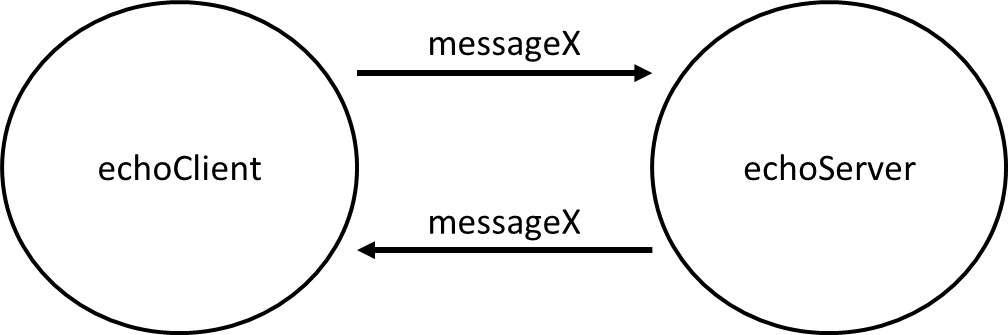
This is your first go at creating a client-server system. The objective is to get it working on the same machine and then on two different ones.

See the appendix for how to run a client and a server in separate console windows in Eclipse.

# What to do

The following diagram shows the networked application.

echoClient and echoServer are two processes (programs) that interact via the Socket API. You can run them on the same computer or on different computers (see later).



echoClient sends a message “messageX” to echoServer. echoServer then sends it back. i.e. it’s an echo! The protocol sequence (the exchange of messages) is therefore…

|  |  |  |
| --- | --- | --- |
|  | **echoclient** | **echoserver** |
|  |  | [run echoServer] |
|  | [run echoClient] |  |
|  |  | [accept echoClient connection] |
|  | WHILE NOT TERMINATED | WHILE NOT TERMINATED |
|  | READ “messageX” FROM user |  |
|  | SEND “messageX” TO echoServer |  |
|  |  | Receive “messageX” from echoClient |
|  |  | Send “messageX” to echoClient |
|  | RECEIVE “messageX” FROM echoServer |  |
|  | PRINT “message X” |  |
|  | ENDWHILE | ENDWHILE |

Note that a protocol is just the exchange of messages and their format. […] represents an action that takes place in support of this and is included for clarity.

The algorithms for echoclient and echoserver are…

echoclient

WHILE NOT TERMINATED

READ messageX FROM user

SEND messageX TO echoServer

RECEIVE messageX FROM echoServer

PRINT messageX

ENDWHILE

echoserver

WHILE NOT TERMINATED

RECEIVE messageX from echoClient

SEND messageX to echoClient

ENDWHILE

Now try getting the following code running on the same machine (**localhost**). Note that the code is also on BBL.

# echoserver

import java.net.\*;

import java.io.\*;

public class echoserver {

public static void main(String[] args) throws IOException {

//Get the id of the local machine – not critical – included here just to show you how

//Declare an object to store your computer's name

InetAddress computerAddr = null;

//Now store the local computer's name

try{

computerAddr = InetAddress.getLocalHost();

}

catch(UnknownHostException e){

System.out.println(e);

}

//Now print it out to the screen

//You will need to use this number in your client program

System.out.println("The address of this computer is... " + computerAddr.getHostName());

//Now set up the server socket on port 4000-4999 on the local machine

//Change the port number 4321 to anything in that range (unlikely there will be another process

//running on that port)

ServerSocket serverSocket = null;

try {

serverSocket = new ServerSocket(4321);

} catch (IOException e) {

System.err.println("Could not listen on port: 4321.");

System.exit(1);

}

System.out.println("Echo server up and waiting");

//Wait for client connection. When a client connects, make the link and carry on

Socket clientSocket = null;

try {

clientSocket = serverSocket.accept();

} catch (IOException e) {

System.err.println("Server socket failed.");

System.exit(1);

}

// Connect the input and the output to and from the socket

PrintWriter echoOutput = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader echoInput =

new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

String inputLine, outputLine;

//Repeatedly loop getting input from the client and just send it back

while ((inputLine = echoInput.readLine()) != null) {

outputLine = inputLine;

echoOutput.println(outputLine);

}

}

}

# echoclient

import java.io.\*;

import java.net.\*;

public class echoclient {

public static void main(String[] args) throws IOException {

//Set up the socket, in and out variables

Socket echoSocket = null;

PrintWriter echoClientOutput = null;

BufferedReader echoClientInput = null;

//Localhost is the generic name of the computer that your echoserver is running on

//Change port 4321 to the port that your server is connected to

//The code then connects the input and output

try {

echoSocket = new Socket("localhost", 4321);

echoClientOutput = new PrintWriter(echoSocket.getOutputStream(), true);

echoClientInput = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));

} catch (UnknownHostException e) {

System.err.println("Can't find host");

System.exit(1);

} catch (IOException e) {

System.err.println("Couldn't get I/O for the connection to: 4321.");

System.exit(1);

}

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

String fromUser;

String fromServer;

//Type some input, press return and see what comes back

System.out.println("echoclient up and running");

while (true) {

fromUser = stdIn.readLine();

if (fromUser != null) {

System.out.println("Client: " + fromUser);

echoClientOutput.println(fromUser);

}

fromServer = echoClientInput.readLine();

System.out.println("Server: " + fromServer);

}

}

}

# Now try different machines

Now let’s try running on two different machines. One will be the server and the other will be the client. Find the IP address of the machine running the server (use ipconfig in a cmd window or run the server code above (which will print the IP address). You will need to change the client with the server IP address. Once you’ve done this run the server on the designated machine first, then run the client on the other machine. You should now be echoing messages from one machine to another!

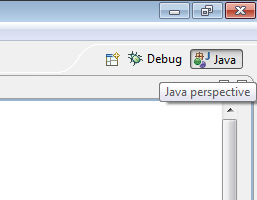
When you have done this, try stepping through each line of your code and discuss with your neighbour what the code is doing.

# Appendix – How to run two (or more) processes in Eclipse

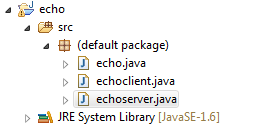
Note you can download Eclipse from [www.eclipse.org](http://www.eclipse.org)

Follow these instructions to run a client-server program in eclipse. This is a little bit different to what you are normally used to. Instead of compiling and running a whole package you are compiling and running a CLIENT and a SERVER separately as TWO SEPARATE PROCESSES linked together by TCP/IP.

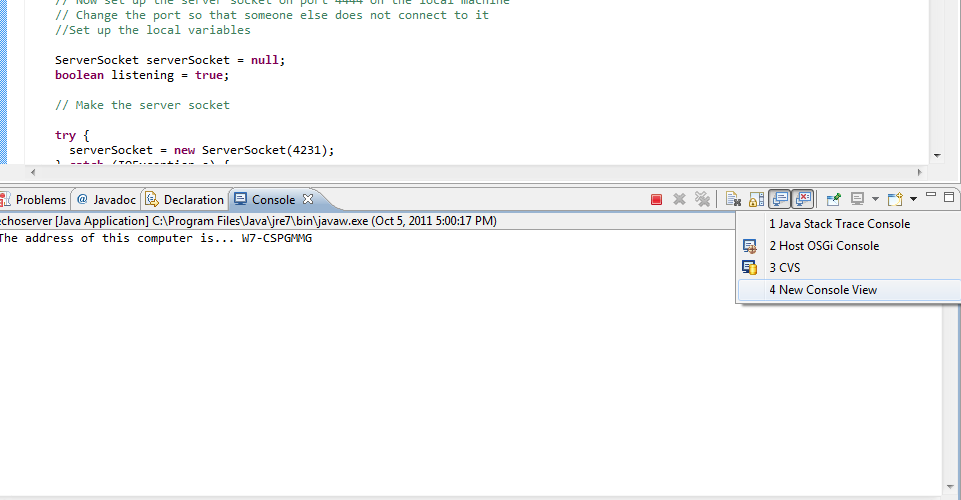
1. Open java perspective



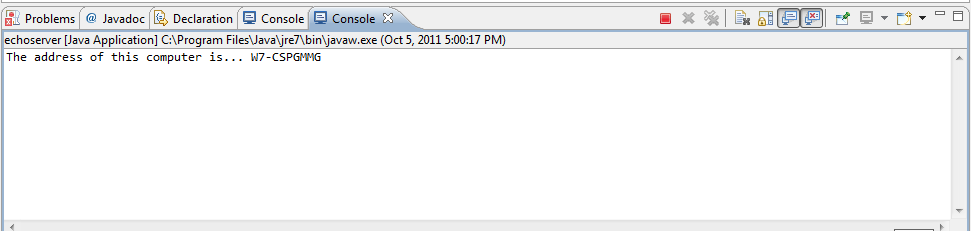
2. Make a new java project and add source codes to it (the ones below are examples – use the codes for the relevant tutorials). Do this by dragging and dropping your files to src (eclipse will automatically add them – select ‘copy’ when asked). Compile/run as appropriate the server code (make sure you’ve got the window with the server code selected and click on the run button).



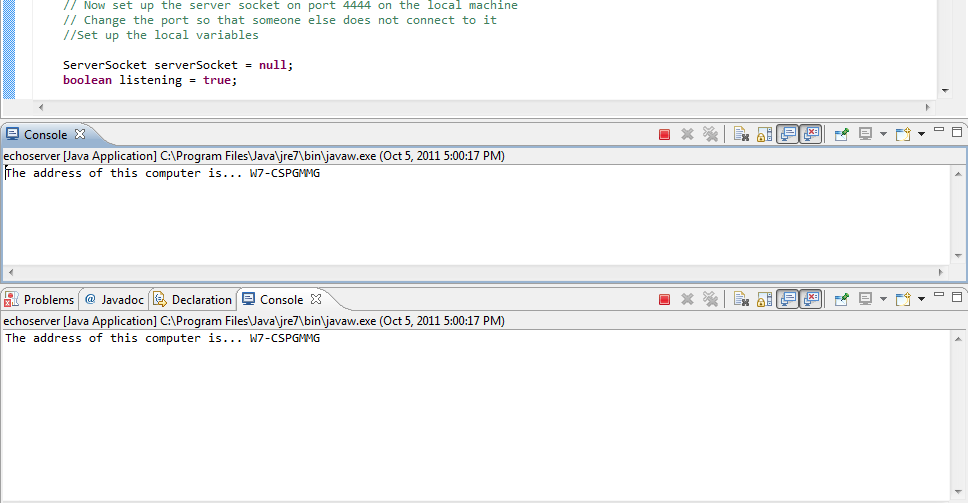
3. Create new Console view (or any number of Console you need for multiple clients)



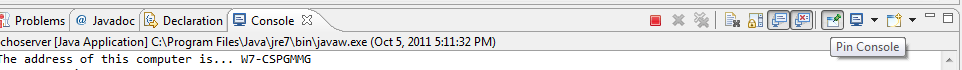
4. Now you can see two Console tabs



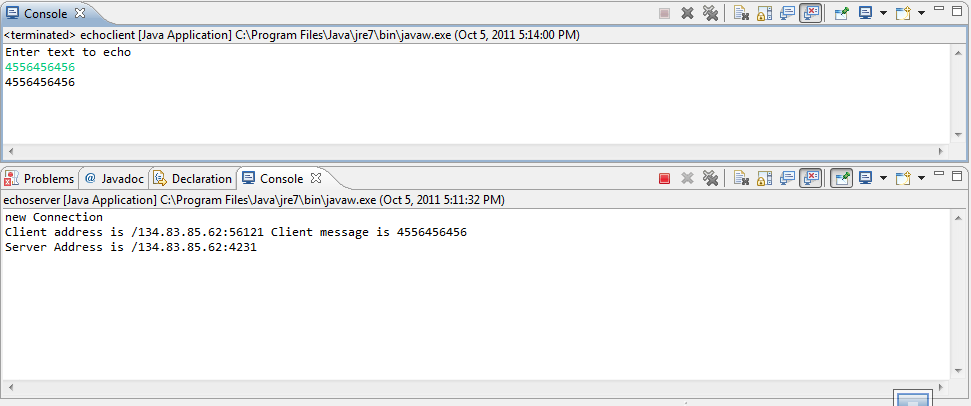
5. Drag and drop one of the console tab anywhere in perspective for better view



6. Pin one of Consoles



Run the client code without stopping the server and enter something appropriate in the console. Make sure you have the client code window selected and then see what happens in the other console (the server’s). What you should see is something along the following lines…



The other running program can be selected from console view



Machine generated alternative text: r
echoclient [Java Application] C:\Program Fil  1 echoserver [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Oct 5, 2011 5:11:32 PM)
to echo rn 2 <terminated> echoclient [Java Application] C:\Program Files’Java\jreJ\bin\javaw.exe (Oct 5, 2011 5:14:00 PM)
@ Javadoc E. Declaration  Console
ea Application] C:\Program Files\Java\jre7\bin’ijavaw.exe (Oct 5, 2011 5:11:32 PM)
: ion
ess is /134.83.85.62:56121 Client message is 4556456456
‘ess is /134.83.85.62:4231
________