TCP Socket Programming

# CSCI 4760

**What is this?** The ***Transmission Control Protocol (TCP*)** transmits a sequence of bytes between two processes over the network. Normally we would expect these two processes to be running on different hosts, but for testing purposes we can send messages between two processes running on the same computer.

A ***socket*** is the programming abstraction (object) through which a program interacts with the TCP network software. Sockets are quite similar to File objects, which are programming abstractions through which a program can read and write bytes on a disk.

**Client-server paradigm**: It is not practical for both sides of the conversation to begin communicating at exactly the same time. In the ***client-server*** paradigm, the server side must begin listening on a known ***TCP port*** before the client side attempts to connect to that port.

**Virtual machine implementation**: You can either write the code on Eclipse or directly on your VM, but it must run on the VM. I prefer the former method, as the ease of auto-generating code and finding syntax errors on Eclipse outweighs the hassle of repeatedly uploading versions of the code.

You can move the code from your local machine in two ways:

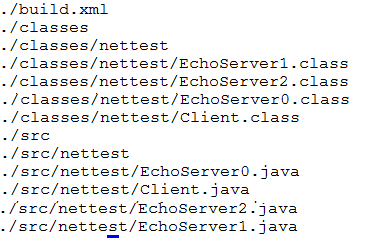
* Direct connection to the Internet address 172.17.152.*xx*, where your machine’s host name is vm*xx*. (The Domain Name Service, which translates alphanumeric host names to numeric Internet addresses, translates the VM addresses only for those machines on the nike sub-network). This should work if you are on the UGA campus network, or connected via a VPN tunnel. Tunneling will be discussed later in this course, but you can establish a tunnel by visiting remote.uga.edu
* If this is not possible, use your SSH/SCP client to upload the files to nike and then hop them over to your vm via scp.

## Setting up ANT:

1. To see whether ANT is already installed on your VM, type

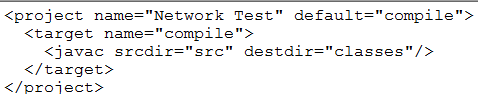
**which ant**

– you get a response “No ant in ….” followed by a list of directories. This list is your search path: the shell will search these directories whenever you enter a command. You can configure your search path via your startup scripts, but the package manager yum can do it for you.

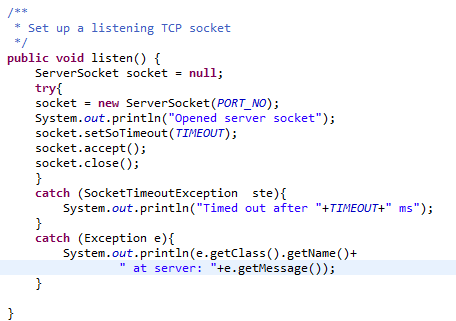
1. Since we don’t have ANT, we need to install it. Here are the commands to do so:
2. The first command searches for the package to install and the second one actually installs it. We had to guess which of the packages whose name contains “ant” is actually the base package. To verify that we have actually installed ANT, enter which ant again.

## Build directory setup

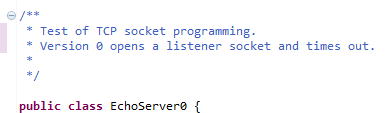
1. Create a project directory **TCP\_test**. Inside TCP\_test, here is the directory layout we will be using. (The package name for the classes in this exercise is nettest.) You need to create the directories **src**, **src/nettest**, and **classes**.
2. Here is the build.xml build file for this exercise. Because we will be running two Java classes at the same time, we will run them from the command line.[[1]](#footnote-1)



## Creating a listening server

Here is the listen method of the first server class, EchoServer0.java. This file goes in src/nettest.

1. Please take a moment to examine this code, as it contains some features of interest:

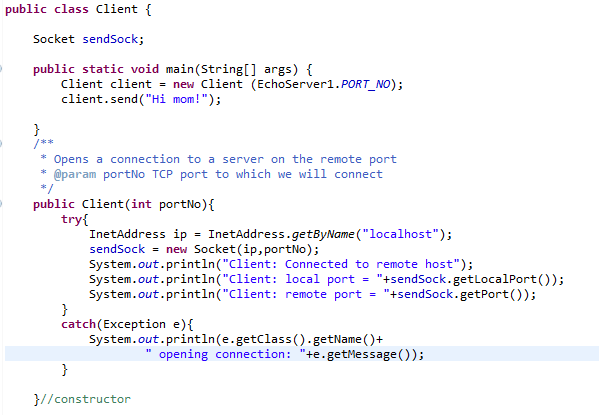
* Take a quick look at the Javadocs for the ServerSocket class. This class is used to listen for an incoming connection request. When the request arrives, the client and server will each created a Socket object to represent one end of the established connection.
* The PORT\_NO and TIMEOUT parameters will be set as class-level static variables. Since the timeout value is in milliseconds, I suggest 2000 for the TIMEOUT and 2013 for the PORT\_NO.
* When we open a ServerSocket for listening, the code will block until either a remote client makes a connection request or the timeout occurs. Note the two catch clauses – the first one handles the SocketTimeout event, which is the expected behavior for this example. If an Exception occurs and is not the SocketTimeout event, we don’t know what it is! The expression  prints out the specific class name of the Exception e.

1. Add a class-level comment as shown here. Write a main method which instantiates an EchoServer0 object and calls its listen method. (If you are coding in Eclipse, there is a checkbox in the file creation wizard to create a main method.)
2. If you are coding in Eclipse, upload EchoServer.java to the src/nettest directory.
3. Compile the class by typing **ant**. Edit and repeat until the compilation succeeds.
4. Enter **find** to see what has happened. You will see that ANT has created a directory classes/nettest which now contains the class file.
5. To run the class “by hand”, enter **java –classpath classes nettest.Echoserver0**. The ‘-classpath’ argument is needed because Java does not know where to find the class file. Notice that Java translates the package name, nettest, into a subdirectory of classes. If you are successful, you should see a message that the server has opened a socket, followed a couple of seconds later by a message that the socket has timed out.

## Creating a TCP connection

1. Create a new class EchoServer1 by copying the EchoServer0 code. This server will actually receive and print a message from a client, so modify the class level comment appropriately. Add the following code in the listen method:



1. Here we introduce a Socket object to represent the established connection between client and server.
2. Upload this file if necessary, then run ant to compile. (Note that the javac task compiles all uncompiled Java classes in the src directory; it will not recompile EchoServer0, since this class has not been changed.
3. Run EchoServer1 by typing **java –classpath classes nettest.Echoserver1.** You should get the same two messages, since there was no connection attempt.
4. Creating a Client class
5. Here is the code for the first client class:
6. 
7. Check out the InetAddress class, which represents the IP address of the machine.
8. TO BE CONTINUED…

1. If you are really eager to automate this test, look up the ANT parallel task for running multiple simultaneous threads. [↑](#footnote-ref-1)