Overview. We have now seen how to use the `Socket` and `ServerSocket` classes to implement simple protocols that involve processing a single line of text. This week you will use these classes, plus a few more, to implement a more complicated protocol. In addition, you will properly catch and handle some network-related exceptions, including socket timeouts.

A Simple Protocol. You will have, as before, a client that opens up a `ServerSocket` and accepts connection requests from it. However, your server should ask the user for a port number (rather than having the port number “hard-wired” into the program).

The server will repeatedly “accept” connections from a client and, for each such connection, will do the following:

- immediately send the client a string containing the name and IP address of the server (use the `getLocalHost()` method of the `InetAddress` class)
- receive a sequence of numerical strings from the client, followed by the string “stop”; no acknowledgement is sent for the numerical strings
- after the “stop” command is received, return a string containing the average of the values received (should be the string representation of a `double` value) and wait to accept another connection request

If an entire minute passes with no connection requests received, the server should close its server socket and halt. This is achieved by using the `setSoTimeout()` method of the `ServerSocket` class and “catch”-ing a `ServerSocketTimeout` exception. Furthermore, if there is a delay of more than ten seconds between successive inputs from the client, the server should print an error message and return to waiting for another request to connect. This is achieved by using the `SetSoTimeout` method of the `Socket` class and either “catch”-ing an appropriate exception (such as the `NoSuchElementException` while trying to read from the client socket) or by using some other methods from the relevant classes.

The server should use “try ... catch” constructs around the `ServerSocket` constructor to gracefully catch any `IllegalArgumentException` that occurs as a result of trying to connect to a port number that is outside the legal range. (This is kind of kludgy — wouldn’t it make more sense to check the range before invoking the constructor? — but I wanted you to have to use `try` and `catch` here.) Note: I am not asking you to check for things like illegal characters in the user’s input. Some exceptions, such as accidentally handling non-numerical data, or trying to take the average of an empty list, or other things, may be handled by the default exception-handling facilities, even if they cause the program to crash. I’m interested in catching socket-related exceptions such as timeouts, invalid host, invalid port, etc.

The client should
• prompt the user for a host name and a port number for the server,
• connect to the server (with appropriate graceful handling of problems in connecting),
• receive from the server and print the name and IP address of the server,
• prompt the user for a sequence of numbers, ending with “stop”
• print the average returned from the server

It should be possible to re-run the client any number of times in succession as long as successive sessions are less than a minute apart. During a single session, if more than ten seconds elapse between two inputs from the user, the server socket will disconnect, and your client should somehow detect this and gracefully halt.

The next page shows some sample sessions (the left and right sides are not supposed to match).
Server:

$ java TCPServer
Enter server port number: 123456
Port out of range:123456; exiting program

$ java TCPServer
Enter server port number: 12345
Client taking too long; closing
Socket timed out; exiting program

$ java TCPServer
Enter server port number: 12345
Client taking too long; closing
Socket timed out; exiting program

Client:

$ java TCPCClient
Enter server hostname: alden27
Enter server port: 12345
Host alden27 not recognized; ending program

$ java TCPCClient
Enter server hostname: aldenv27
Enter server port: 123445
Port 123445 out of range; ending program

$ java TCPCClient
Enter server hostname: aldenv27
Enter server port: 12345
Connected to server aldenv27/141.195.226.27
enter numbers to be averaged:
3.4
5.68
10.993
-16.7
0
23.12
-8.88
stop
FROM SERVER: avg = 2.516142857142857

$ java TCPCClient
Enter server hostname: aldenv27
Enter server port: 12345
Connected to server aldenv27/141.195.226.27
enter numbers to be averaged:
10
.... long pause ....
stop
Server timed out; halting program