INFO 341 Project TWIT - Part 1

Due: Friday November 2, 2012 by 10:00pm

During this project you will be building a peer-to-peer version of a microblogging application. An example of microblogging is Twitter. Over the next few weeks you will build something similar – where one individual can elect to "follow" the status updates or posts of other individuals.

For Part 1 of this project you will build a very basic peer-to-peer chat system that will give you mastery over creating network server and client connections. Part 1 has been broken into four tasks. When you have completed all four tasks then you should have a working solution to Part 1. The remaining parts of this project will be built off of your Part 1 solution.

Part 1, Task 1 – Modify Your Simple Console

Start this task with the simple console you build for Lab Assignment 1.

Modify your console to implement a "`connect`" command. This command will be used to make a connection to a remote machine. Your console should be able to parse the command "`connect foo.bar.com:5600`" to separate the "`connect`" command a machine hostname "`foo.bar.com`" and the protocol port number "`5600`". Here in Task 1, when you see the "connect" command, your console should print something like, "`connecting to: <hostname>:<port>`" where the string "`<hostname>`" is the name of the host identified by appropriately parsing the command, and "`<port>`" is the protocol port identified by appropriately parsing the command.

Also, to be useful later your Console class needs a method that you should call "message()" that takes two parameters, a String that is a message and a String that represents who is sending the message. The method will need to write the message string and who it is coming from on System.out.

Part 1, Task 2 – Building a Connection

A "Connection" is the instantiation of a socket that represents the way to communicate to a remote machine. A Connection is a wrapper around a Java Socket that makes it easier to send/receive (or write/read) the socket.

For this part you should write a Java Connection class that extends Thread. You should be able to instantiate the Connection class in two ways. The first way to instantiate the connection class is when you are opening a new connection to a remote machine. That form minimally needs a hostname and a protocol port number in order to create a new outgoing connection to a remote machine. To instantiate that version of Connection you would have a line of code like:

```
Connection c = null;
c = new Connection("foo.bar.com", 5600);
```

This version of a Connection is used by client applications when they connect to a server. However, a Connection needs to work for servers as well. The second way you need to instantiate your Connection class is by using an existing Socket. Servers that accept incoming connections generate a Socket object when the incoming connection is created.

```
Socket incoming_socket = server.acceptConnection();
Connection c = new Connection(incoming_socket);
```

This second version of a connection is used by a server to receive incoming connections, and then communicate with that client through the resulting Connection.

This Connection class needs three additional methods to be useful. The first method that connection needs to implement is the "`run()`" method. This method is inherited from the Thread class, and allows your connection to watch for the incoming communications using the incoming Stream class provided by the Socket. The second method it needs is a method called "send" which takes a String and writes it to outgoing Stream class provided by the Socket. Your Connection class also needs to implement a "`close()`" method that closes the connection, closes the Socket, and terminates the thread.

Part 1, Task 3 – Building a Server

A server has a very simple job. A server listens for incoming connections from a client and when that connection is made it instantiates your Connection class and hands that connection off to some other part of your program.

For this part write a Java Server class that listens on a port for an incoming connection. When it receives that connection have the Server class instantiate a Connection class using the Socket that it receives. Your server class should also be an extension of the Thread class. Because server inherits from Thread, you need to implement the "run" method. Your run method should be a loop that waits for incoming connections, instantiates the connection class, and hands that connection to some object that will communicate through the connection.

Part 1, Task 4 – Building A Simple Peer-to-Peer Chat

With these three classes Console, Server, and Connection you can now build a very simple peer-to-peer chat application. Most of the work is adding code to your Console class.

Add a variable to your Console class to store a Server class.
Add a variable to your Console class to store a Connection class.
Modify the main loop of your Console so that when the user types the "connect" command, you create an outgoing peer connection to the specified host.

Modify your Console main loop so that if a peer Connection is present any words, that are *not* commands, typed at the command line are written/sent to the peer connection. That is, if a connection is present, then there are no unrecognized commands. Instead an unrecognized command is text to be sent to the remote client. As a side effect, you cannot send lines of text to a remote client if those lines of text start with any of the command words.

At this point your Console class should implement the following commands:

`connect` *`<hostname or IP address>:<port>`* - connect you the specified peer (host) running the same version of your program and listening on the same, specified protocol port

`halt` - Halt the console and exit cleanly with no errors.

`time` - Print the current wall clock time in the format HH:MM:SS [AM/PM].

`date` - Print the current date in the format MM/DD/YYYY

`help` - Print an explanation of the commands that it knows

Turning In

Your Java code should compile and run correctly from the command line. If you use a development environment, you should make certain that your code will compile and run from the command line.

All of your Java classes should be in a `package` that is named by your lastname and the last three digits of your student ID. So if your last name is "McStudent" and the last three digits of your student ID were "820" your code would include a statement like:

```
package mcstudent820;
```

If you have your Java CLASSPATH variable set correctly you would be able to run your Console from the command line by typing:

```
java mcstudent820.Console
```

You will turn in your Java source files as one Zip file. Since packages are assumed to be in directories that are named the same as the package (e.g., in the example above, your code would be in a directory named "mcstudent820") you can just Zip that directory and submit the single file through the course Catalyst Dropbox. Absolutely no assignments will be accepted through email.