CISC 3120 Fall 2012 Homework 9

Instructions

- This assignment **is due by midnight on December 12th** and you will submit this online — instructions below:
  1. Create a ZIP archive from the `.java` files that make up your homework.
  2. Go to:

     `http://agents.sci.brooklyn.cuny.edu/parsons/cisc3120/`
  3. Login and submit your ZIP archive.
  4. You can submit as many times as you like before the deadline—only the final version of the homework will be graded.
  5. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions. . . (since it can make it a lot harder for me to grade your work — grrrr!)

1 Overview

For this homework you will write a client/server pair that create a simple chat application, allowing lines of text to be passed from one machine to another.

You may find the `SimpleServer` code a useful starting place. You can download this from the class webpage.

2 Simple chat

1. You should write a server class, called `ChatServer`, and a client class called `ChatClient`.
2. `ChatServer` should take as an argument a port number, and listen on that port.

   If no port number is provided, the server should use a sensible alternative.

   This should be documented in the code.
3. `ChatClient` should take as an argument an IP address and a port number, and try to open a connection to the port at that address.

   If no port or IP address is provided, the client should fail gracefully.

   This should be documented in the code.
4. `ChatClient` should allow a user to type in a line of text, and, pass this text to `ChatServer`.
5. Similarly, `ChatServer` should allow a user to type in a line of text, and, pass this text to `ChatClient`.
6. Both `ChatServer` and `ChatClient` should be able to terminate the connection at any time.

   This termination should not lead to any error messages from either `ChatServer` or `ChatClient`.

3 Run simple chat over the network

1. You should be able to test `ChatServer` and `ChatClient` locally, that is running both programs on the machine you use for homework.

   (This is certainly true if you are running a Unix-based system like OSX or Linux, and should be possible if you are using Windows).

2. If not, you can upload `ChatServer` to your account on the department machine, the one you used for the last homework, and connect to it over the internet.

3. To run it, you will type:

   `java ChatServer <port number>`

   where `<port number>` is the number you were assigned in class. If you don't have a port number, contact me.

4. Then on your local machine you need to run `ChatClient`, giving it the IP address (which you already know) and the port number.

5. Even if you can run client and server locally, you should test uploading `ChatServer` and running it like this because that is what I am going to do when I mark the homework.

4 Chat GUI

Now you are going to build a GUI for the client. The GUI should have:

1. A text field into which the user types their message.

2. A text area (with scroll pane) that displays the text the user types in the client and the text that arrives from the server.

3. A menu option for opening a connection to the server.

   This will need to ask the user for the IP address and port. By now you should be able to figure out how to use a dialog to do this.

4. A menu option to close the connection to the server.

5. A `Help` menu item that explains how to use the client.

This code will likely need to be broken into several files. Call the top-level file — the one that needs to be executed to run the GUI — `GUIClient`.

You should also include a file `README` that explains which of your `.java` files are used by `GUIClient`. (Remember, I'm going to be unpacking a number of files, you need to explain what they all are).

5 Connect GUIClient to ChatServer

1. `GUIClient` should be able to invoke the relevant functionality from `ChatClient` that made `ChatClient` able to pass messages with `ChatServer`.

2. In other words, you should be able to run `ChatServer` on the department machine, run `GUIClient` on your local machine, and pass messages between the two.

3. As for the original chat programs, I will test the operation of your code by running `GUIClient` locally and having it communicate with ChatServer on the remote machine. You would be wise to test it under the same circumstances.

6 Challenge problem

1. As it stands, the chat program is a bit unrealistic — it requires one party to run the server and then have the other party connect to it.
2. A more realistic service would run a server at some fixed location, and allow clients to connect to it and then communicate.
3. For additional credit, write such a program.
4. You'll need a server that listens on two ports for the clients, and then reads from one and passes what it reads to the other.
5. The client you developed for the previous part shouldn't need much modification (if any) to work with this client — it is just reading from and writing to a socket connection on a port.
6. As a result, you should be able to run two copies of `GUIClient` that allow users (who might be on different machines) to communicate.
7. To test this over the network, you will need a second port number for the department machine that you run the server on. Contact me to get this. (Don't just grab another port since that will likely collide with someone else).
8. As for the `GUIClient` you'll need to include a(nother) `README` that explains which files are used for this part of the homework.

7 What to submit

The ZIP file you submit should include:

1. `ChatServer.java` and `ChatClient.java` from the first part of the homework.
2. All the `.java` files used by `GUIClient`.
3. Your `README`.
4. Any files you write for the challenge problem.
5. Remember you can only submit one ZIP file through the submission system, so everything needs to be in there.
6. If you want to check you have included everything, you can upload your ZIP file to the department machine, unpack it, and try compiling the code. That should identify any missing files.

8 How I will mark your work

1. You will get credit for each of the things listed above that you have implemented and which work as I have described them.
2. You will also get credit for the **way** in which you do this. That is programs that do not adhere to what I consider to be good programming style will not gain as much credit.

   You can get an idea of what I consider to be good style by looking at the code I have given you.
3. To get ful credit you will also have to document your code. Again looking atmy code will give you an idea of what I am looking for.

4. Also note that your program **must compile and run** for you to get any credit at all.

   (At this point in your computer science career it is unacceptable to submit programs that don't compile and run).

   Bear in mind that I will be using Java version 1.6 to assess whether your code compiles and runs.

5. Thus it is better for you to submit a program that works and contains less functionality than it is to submit a program with more functionality that does not work.

6. You can get full credit for this homework **without** solving the challenge parts.