School of Computing, University of Leeds

XJCO2221 Networks

Coursework: Client and multi-threaded server

Deadline: 10am, Saturday 30th April

For this coursework, you will implement client and server applications for a simple online system that allows users to sign–up to one of a number of lists, something like the sign-up feature provided by Minerva. The number of lists, and the maximum number of members for each list, is specified when the server is launched. The client can then request information about the current totals, the members of a specified list, and request to add a name to a specified list.

Important: Your submission must follow the requirements under 'Submission'; in particular:

- Both the client and the server must **only** consider arguments provided on the shell command line – **neither** should expect interaction from the user after they are launched;
- You should first check your submission format by submitting to **Coursework: Check submission** (under *Minerva→Submit my work*), which will run a series of checks. Only make the actual submission once all of these checks have been passed.

Requirements

To get started, download the file `cwk.zip` from Minerva and unarchive it using `unzip cwk.zip`. You should have a directory `cwk` containing the following files and subdirectories:

```
cwk --- client --- Client.java
     |
     -- server --- Server.java
```

Empty class files for the client and server have been provided. **Do not change the names of these files**, as we will assume these file and class names when assessing. You are free to add additional `.java` files to the `client` and `server` directories but your submission must work when the procedure described below is followed.

The requirements for the **server** application are as follows:

- Accept the number of lists, and the maximum number of members per list, as command line arguments when launched. All lists should be empty initially.
- Run continuously.
- Use an `Executor` to manage a fixed thread-pool with 25 connections.
- Store the members of each list, where each member is represented by a single `String`.
- Return information about the lists, and attempt to add new members to a list, when requested by a client (see below).

- Create the file `log.txt` on the `server` directory and log every client request, with one line per request, in the following format:
  `date|time|client IP address|request`.
  Nothing else should be output to the log file.

The requirements for the **client** application are as follows:

- Accept one of the following commands as command line arguments, and performs the stated task in conjunction with the server:
  - `totals`: Displays the number of lists, the maximum per list, and the current number of members of each list.
  - `list n`: Display every member in the given list, one member per line.
  - `join n name`: Attempt to add a member `name` to list `n`, and return to the user `Success` if this was successful, or `Failed` if not.
- For the `join` command, full names with spaces should be enclosed in quotes so that they are parsed as a single `String`; see examples below.
- Quits after completing any one command.

Your server application should listen to a port number in the range 9000 to 9999, but otherwise you are free to choose the port number. Both the client and the server should run on the same machine, *i.e.* with hostname `localhost`.

Lists are index by number starting from 1; so if there are 3 lists, the client should reference them as lists 1, 2 and 3, *not* 0, 1 and 2.

All communication between the client and server must use sockets. Your solution must use TCP, but other details of the communication protocol between the server and the clients are not specified. You are free to devise any protocol you wish, provided the requirements are met.

Neither the client nor the server should expect interaction from the user once they are executed. In particular, instructions to the `Client` application **must** be *via* command line arguments. In the case of an invalid input, your client application should quit with a meaningful error message.

Your code should adhere to the Java coding standards described in `JavaCodingStandards.pdf` on Minerva.

Example session

Suppose you want 3 lists, each with a maximum of 2 members. Then you would `cd` to `cwk/server` and launch the server using

```
> java Server 3 2
```

Now in another tab or shell, `cd` to `cwk/client`. If you execute the following commands, the output should be something like that shown.

```
>java Client totals
There are 3 list(s), each with a maximum size of 2.
List 1 has 0 member(s).
List 2 has 0 member(s).
List 3 has 0 member(s).

> java Client join 1 "William Haydon"
Success.

> java Client join 1 "Constance Sachs"
Success.

> java Client join 3 "Gerald Westerby"
Success.

> java Client join 1 "Percival Alleline"
Failed.

> java Client list 1
William Haydon
Constance Sachs

>java Client totals
There are 3 lists with a maximum size of 2.
List 1 has 2 member(s).
List 2 has 0 member(s).
List 3 has 1 member(s).
```

Note your application does not need to follow *exactly* the same output as in this example, as long as the requirements above are followed.

Guidance

You will need the material up to and including Lecture 11 only to complete this coursework.

You may like to first develop `Client.java` and `Server.java` to provide minimal functionality, following the examples covered in Lectures 7 and 8. You could then add another class that handles the communication with a single client. This will make it easier to implement the multi-threaded server using the `Executor`. Multi-threaded servers were covered in Lectures 10 and 11. You will need to use input and output streams; these were covered in Lecture 6.

How you store the lists on the server is not specified — note you are not required to use a database — so you are free to choose whatever means you like. `Java` has a broad range of container classes which you may like to consider. For this coursework, it is recommended you choose whatever is the most convenient solution for you.

Note that although you can imagine much more functionality that a real sign-up system should provide, such as checking the same person is not assigned to multiple lists, or allowing users to leave a list, the requirements above are sufficient to assess your grasp of networking in Java, and are all that are required for this coursework.

Learning objectives

- Implementation of client and multi-threaded server applications.
- Design of a communication protocol to meet the specified requirements.
- Use of sockets to transfer information between server to client.

Marks

This coursework will be marked out of 30.

7 marks	:	Basic operation of the `Server` application, including use of thread pool and log file output.
4 marks	:	Basic operation of the `Client` application, including meaningful error messages
12 marks	:	Correct functionality of `totals`, `list` and `join` commands.
7 marks	:	Good structure and commenting. Adherence to the Java coding standard provided.
Total: 30		

Submission

Remove all extraneous files (*e.g.* `*.class`, any IDE-related files *etc.*). You should then archive your submission as follows:

1. `cd` to the `cwk` directory
2. Type `cd ..`
3. Type `zip -r cwk.zip cwk`

This creates the file `cwk.zip` with all of your files. Make sure you included the `-r` option to zip, which archives all subdirectories recursively.

To check your submission follows the correct format, you should first submit using the link *Coursework: Check submission* on Minerva (under *Submit my work*). Only once it passes all of the tests should you then submit to the actual submission portal, *Coursework: Actual submission.*

The following sequence of steps will be performed when we assess your submission.

1. Unzip the `.zip` file.
2. `cd` to `cwk/client` directory and compile all Java files: `javac *.java`
3. `cd` to `cwk/server` directory and do the same.
4. If there is a `log.txt` file on the `server` directory, it will be deleted.
5. To launch the server, `cd` to the `cwk/server` directory and type `java Server 3 2`
6. To launch a client, `cd` to the `cwk/client` directory and type *e.g.* `java Client totals`
7. Multiple clients will be launched.

If your submission does not work when this sequence is followed, you will lose marks.

Disclaimer

This is intended as an individual piece of work and, while discussion of the work is encouraged, what you submit should be entirely your own work. Code similarity tools will be used to check for collusion, and online source code sites will be checked.