School of Computing, University of Leeds

COMP2221
Networks
Coursework 2: Client and server architectures

Deadline: Monday March 23rd, 10am.

If you have any queries about this coursework, visit the Yammer page for this module (`COMP2221 1920`). If your query is not resolved by previous answers, post a new message.

Learning objectives

- Implementation of a client and a multithreaded server.
- Design of a communication protocol to meet specified requirements.
- Use of sockets to transfer data over a network.

This piece of work is worth **25%** of the final module grade.

Task

Write two applications, a **client** and a **multi-threaded server**, such that the client allow users to vote in a poll that is managed by the server.

As an example, suppose the poll has three options to vote for, `apple`, `pear` and `orange`. Initially each option has 0 (zero) votes. When a user calls the client to vote for one of the options, the client contacts the server to cast a vote for that option, and the server increases the number of votes for that option by 1. The client can also request to see the current state of the poll, *i.e.* the current number of votes for each option, which should be returned to the client and displayed to the user. For the purposes of this assessment, the same client can cast as many votes as it likes.

Specifically, the **client** application should:

- Accept the command line arguments `show`, or `vote <option>`.
- For `show`, it should request the current state of the poll from the server, and display it.
- For `vote <option>`, it should request that the server increases the vote count for `<option>` by 1, and display the message returned by the server.
- Quits after each command.

The **server** application should:

- Accept at least two options to be voted for, as command line arguments when launched, *e.g.* `java Server apple pear orange`
  The server should quit with an error message for less than two options.
- Run continuously.
- Use an `Executor` to manage a **fixed** thread pool of size 20.
- If the client makes a vote for `<option>`, the vote count for `<option>` should be increased by 1, and a message returned to the client stating the new total.
- Conversely, if the option does not exist, an error message should be returned instead.

- Following a request by a client, return the current state of the poll with one line per option, where each line contains the option and the current count. A possible output is:
  `'apple' has 1 vote(s).`
  `'pear' has 0 vote(s).`
  `'orange' has 3 vote(s).`
- The server should log every request in a local file `log.txt` with the format
  `date:time:client IP address:request`
  There should be exactly one line per request, and no other information should be output to the log file except that given above.

The listening port should be `7777`. Although you should test and submit your solution with both the client and server running on `localhost`, it should in principle work across a wide area network. Therefore the client and server should only communicate *via* sockets.

You must use TCP, but the other details of the communication protocol (*i.e.* the messages that are actually sent between the client and server) are up to you, as long as the requirements listed above are met.

Your code should generate meaningful error messages for common problems and user errors.

Your code should adhere to the coding standards in `JavaCodingStandards.pdf` on Minerva. Your solution should expect no interaction with the user other than *via* command line arguments.

Guidance

The material required for this coursework was covered in Lectures 6 to 11 inclusive.

Download the file `cwk2.tar.gz` from Minerva and unarchive it using `tar xzf cwk2.tar.gz`. You should now have a directory `cwk2` containing the following files and subdirectories:

```
cwk2 --- client --- Client.java
      |
      -- server --- Server.java
```

This provides you with empty class files for the client and server. **Do not change the names of these files**, as we will assume these file and class names when assessing (see below). You are free to add additional `.java` files to the `client` and `server` directories but should not create extra directories.

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 can 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`. You can then add functionality one item at a time, *i.e.* each of the `show` and `vote` options. The logging can be left until the end.

Input and output streams, and how to chain them, were covered in Lecture 6. It is not recommended that you chain multiple times from the same stream (*e.g.* chain two output streams from `socket.getOutputStream()`), as this can cause ill-defined and non-reproducible behaviour. Depending on your choice of stream. you may need to manually `flush()` when sending data.

Marks

This coursework will be marked out of 25.

3 marks	:	Basic operation of the `Client` application.
7 marks	:	Basic operation of the `Server` application, including multi-threading and log output.
3 marks	:	Correct implementation of the `show` command.
6 marks	:	Correct implementation of the `vote <option>` command.
6 marks	:	Good structure and commenting. Adherence to the Java coding standard provided.
Total: 25		

Submission

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

1. `cd` to the `cwk2` directory
2. Type `cd ..`
3. Type `tar czf cwk2.tar.gz cwk2/*`

This creates the file `cwk2.tar.gz` that you should submit *via* Minerva.

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

1. Unarchive the `.tar.gz` file.
2. `cd` to `cwk2/client` directory and compile all Java files: `javac *.java`
3. `cd` to `cwk2/server` directory and do the same.
4. To launch the server with some options to vote for, `cd` to the `cwk2/server` directory and type *e.g.* `java Server a b c d`
5. To launch a client, `cd` to the `cwk2/client` directory and type *e.g.* `java Client show` or `java Client vote a`
6. Multiple clients will be launched simultaneously.

Your submission must work when this sequence is followed.

All testing will be performed on a School Linux machine, similar to those in DEC-10.

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.

The standard late penalty of 5% per day applies for work submitted after the deadline.

Ensure you retain the receipt for your submission as you may be asked to produce it at a later date in the event of a disputed submission time.