Internet Computing Open Assessment 1: Web Services

For this assessment, you are being asked to build a well-engineered chat application, similar to the ones built in the module using sockets and RMI. However, your application should use SOAP Java Web Services.

1.1. Task Specification 1: BSc

You need to write two small Java programs: a server application and a client application.

1. When new clients connect to the server they should be able to specify a *name*. Exactly how you go about this is up to you. But remember that this is not GUI course, and you are encouraged *not* to build solutions with complex GUIs. That means you should use a *simple* method of specifying the name.
2. When the server propogates a message to the connected clients, it should include the name of the client the message came from, and the clients should display this information.
3. Clients should be able to leave the server. This should not potentially cause problems - which is the case with the very simple Web Service application in the module notes. *Please not* this is *not* simply a matter of replacing the empty `leave` method in the example!
4. The server should pay proper attention to synchronisation first and performance second. That is, solutions that are inefficient *but correct* are preferable to ones that are efficient but incorrect. Of course, both correct and efficient are better still!
5. Clients should be able to specify the name of another client to whom they wish to send a message *privately*. The next *single* message they send should go *only* to that named client. Specifying the name of a client to chat privately that does not exist should not be a terminal error. Any later messages should go to all clients unless they are specified to be private again.

1.2. Task Specification II: MSc

In addtion to the above, students following the CS-M58 module must add the following functionality.

1. It should be possible for a client to change to *administrator status*. To confirm this, the client must enter the password, which should be 'epidauros'. This does not need to be changeable, and there will be no penalty for sending it in plain text. However, it would be better if it was stored in a configuration file rather than embedded in the source code.
2. Once logged in as the administrator, the client should be able to request a *list* of connected users.
3. There should be a way for a client to *log off* as the administrator.

1.3. Overview and Suggestions

This section points out some issues that you might have to consider, and which might be useful.

- **Doing More.** You may feel that this coursework is too easy. You are welcome to add more functionality but *there is no explicit extra credit for doing this.*
- **Fairness.** The socket and RMI examples are correct in the sense that they do not corrupt the data structures they use. However, they are not strictly *fair* because new clients joining and old ones leaving can be delayed indefinitely by other clients talking. Solutions that address this get (some) extra credit - see assessment below.
- **Polling or Callbacks.** You are free to use either polling or callbacks to implement server-to-client communications. However, the difficulties of using callbacks are considerable and so there is no penalty for using polling.
- **Netbeans and Glassfish/Tomcat.** You can build and test your application however you want. However, I very much recommend you either use the command line tools or a combination of Netbeans and Glassfish as demonstrated in lectures.
- **Identifiying Clients.** Because the simple example this is based on uses integers to identify clients, sometimes people think that they need to continue to use integers for that in their version too. *You don't* - and it probably makes things more awkward if you do.
- **Keep the GUI simple.** This is not a GUI course and you will basically get no marks for the GUI no matter how complicated it is. Therefore, you are encouraged to keep it simple (and it's completely acceptable to use the GUI code from the versions in the notes). I would recommend you stick with the format of the

socket and RMI examples, and use *text* commands for the extra functions I am asking for. You don't *have* to do that, and if you choose to do something more sophisticated I certainly won't mark you down for it. *However* be careful, because in the past some GUIs that looked great on Windows had real problems when I marked them on a Mac because important parts - buttons, labels etc., - did not fit inside the GUI's frame, and so I couldn't see them and the application did not work. You have to try quite hard to do this, and code the GUI badly, but it has happened so be careful. (The main thing to do is to avoid *hardcoding* the location of the GUI components, but to use a *layout manager* instead.)

- **My Code.** You are welcome to look at the code examples in the web service, RMI and socket versions of the chat application for help and inspiration. But please modify it as appropriate to your solution. I really don't want to be reading my own comments for example! The only parts of the code that you need not alter are those concerned with the GUI, but even here, please update the comments. This area is now something that's explicitly considered in the marking scheme. You will find the code on Blackboard along with this assignment. *In the past people have sometimes submitted ONLY my code as a solution* - on the grounds that it meets some of the specification and so they should get some of the marks. To be explicitly clear, there are NO marks for doing this, regardless of the Blackboard marking rubric.

Normally it would be bad practice to hard-code the URL of the server into the client code. However in this case you must do so. The URL should *ideally* specify the host `localhost` and port number 8080 - the rest of the URL is up to you.

NOTE: If you use Netbeans for development, you *may* find that 'localhost' does not work, and that you have to use the name of your own local machine instead. That is fine - but if you do that please let me know in the readme.txt (see below) file so I know I need to edit it.

1.4. Submission Guidelines

You will need to submit your coursework electronically via Blackboard.

Your solution should be in a single file that can *either* be a `gzip'ed` and `tar'ed` file or a conventional ZIP file: please read and follow the instructions below (and don't use a RAR file).

I intend to automate some of the assessment process with scripts to check that solutions work (note though that I will still read your code). Therefore, you *must* do the following:

- Include your source code files.
- Include a file containing simple instructions to run your program called readme.txt.
- You may include other files as well if you wish - e.g. class files.
- It is *vital* that you either submit a ZIP file or a gzipp'ed and tar'ed file - do NOT send me anything else (e.g. RAR files) - this will upset me deeply.
- **Do not submit your coursework by email unless I *explicitly* ask you to - I will delete it.** If Blackboard does not work for you for some reason, send me an email with `[submission]` (the word `submission' in square brackets) in the title and I will resolve the problem. But do not send your coursework with your email unless I explicitly ask you to do so.

I strongly recommend that after you create your submission file you copy it to another folder, and unpack it to make sure it actually contains what you think it contains. (This will also prevent you inadvertantly submitting a corrupt file or empty file - which a couple of people do every year.)

1.5. Assessment

Coursework accounts for 30% of CS-338; and 40% of CS-M58. Assessment and feedback will use the Blackboard rubric on the submission page. BSc students will be marked out of 75 (and the mark scaled up to a % by multiplying it by 4/3; MSc submissions will be marked out of 100.

1.6. Common Mistakes

The usual mistakes that are made are:

- Sending a corrupt or empty file - please make sure it unpacks and works properly.
- Ending up with files in a subdirectory (or series of subdirectories) after unpacking that is not related to the Java package names you used.

- Sending the class files instead of the source files - believe it or not it happens.

1.7. Dates

The deadline for this Open Assessment is 11.00 on *Wednesday 25th March 2015*. To conform to University policy, late submissions will get a mark of zero. Under no circumstances will I grant extensions. However, if you believe you have good reasons for not being penalized, you can make a case to the Computer Science Extenuating Circumstances Committee. I will attempt to ensure that grades and comments are available, and posted, by *Friday 1st April 2015*.