G52APR Coursework 2011 – Part 3

Colin Higgins

**Introduction**
See the Coursework documents for Part 1 and Part 2 for background to this part.

**Data Querying (30%)**
Due: 16th December, 16:00

---

1.1 **Requirements**
For this final part you are required to produce a Java sub-system that integrates with earlier parts of the coursework. It should provide a Database Access System (a **server**) that allows clients to connect to it and exchange messages (data) via Java’s Remote Method Invocation (RMI) libraries. The server should allow concurrent connections and queries. If the requirements are not clear then please feel free to interpret them in any way you feel sensible, but please comment on this interpretation in the supporting report.

You may change earlier parts of the system if you feel it is necessary or desirable. Please note many students are struggling by not thinking through the “design” – although it is pretty trivial. Basically you should realize that only one copy of most of these objects is needed. You could even enforce this via a singleton pattern as suggested in the lecture earlier in the term.
The server should implement the methods given in the `DataQueryInterface` interface. You may wish to add extra methods to the `DataRequest` interface to allow your system to make new SQL requests for the data. An alternative is to use the result sets returned from part 2 and process these to extract the data required for the new queries in Java, although this is likely to run more slowly.

The RMI registry should be established in your code to make the code as stand alone as possible. Perhaps get the basic functionality working first, then make the system concurrent and complete? You should ensure you only ever get a piece of data from the web once – subsequent requests should get it from the database. Also you should only go to the web when necessary. You may need to convert the resultset into an object that can be sent via RMI (ie serializable). You could do this using various data structures. In this case please use a Vector of Vectors of String (`Vector<Vector<String>>`) where the first vector contains the rows of the resultset and each inner vector contains the strings of a resultset row in the same order as they are inserted into the database via the `DataStorageInterface`.

1.2 Testing
You should write your own client that contains methods to exercise the DQ sub-system. If you feel further testing is required then do so and explain in your code and in your report. As before, data testing is an important part of software engineering and program development, so the more complete and thorough the testing, the higher the marks that will be obtained again. Your test suite should not normally be interactive (ie not a GUI) but should test those aspects of the system you deem necessary when it is ran.

1.3 Deliverables
- Source code for your data querying sub-system (the server and supporting code, including anything necessary from earlier parts of the coursework) and test frame (a client): as a zip file, please feel free to zip up the entire (Eclipse?) project including imported libraries.
- A short (approx 500 word) report explaining how your system works, what choices you have made and why. The report should be in plain text.
- A runnable jar file of your system and libraries. This is easy to create in Eclipse and other development environments. Perhaps get a friend to test it executes! Include your

Please submit a zip file of your source code, your report and jar files to cw submit:

430) G51APR cw3: Data Querying CODE (cah)
444) G51APR cw3: REPORT (cah)
445) G51APR cw3: JAR Server(cnh)
451) G52APR cw3: JAR Client (cah)
1.4 Feedback
You will be given written feedback on the code and report.

Notes
As I answer email queries about this coursework I will add clarification notes here:

1) Please feel free to use any code from Parts 1 & 2.
2) Perhaps get the system working for non-concurrent access first, particularly if you are finding the work difficult, then make it concurrent.
3) Perhaps test the fundamental connections etc of your code all work on the Schools system, THEN develop the details outside if you wish. Several people have had trouble at the last minute with porting their code to work at the University.
4) I've just realised you need to submit 2 jar files (one for the server, one for you test client).
5) Please ignore the .DS_Store file that is in the zip file – It's Mac rubbish that gets inserted when I zip the files!
6) From my email...
7) There is probably a easier way to do the RMI part of the coursework than the lecture notes show.

Take a look at some simple code at...

http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/hello/hello-world.html

Which shows a nice way of doing it. You need to use your School user name as the binding to ensure it is unique. So in the code from the above website you will need to replace the "hello" with "AAADDA" (where AAADDA is YOUR user name).

i.e. In the server class the line:

    registry.bind("Hello", stub);

should be:

    registry.bind("AAADDA", stub);

And in the client the line:

    Hello stub = (Hello) registry.lookup("Hello");

should be:

    Hello stub = (Hello) registry.lookup("AAADDA");
We have also been having problems with the RMI registry on Bann, so please develop on a lab machine or your own machine where you will need to run the registry yourself (from within the program).