University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Distributed Algorithms and Systems (DAS4)**			
Coursework Number	**COMPSCI 4019**			
Deadline	**Time:**	**16.30**	**Date:**	**29 November 2013**
% Contribution to final course mark	**20 %**			
Solo or Group	**Solo**	✓	**Group**	
Anticipated Hours	**20 hours**			
Submission Instructions	**See 'What to submit' on page 3**			
Please Note: This Coursework cannot be Re-Done				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via
https://webapps.dcs.gla.ac.uk/ETHICS for all coursework
UNLESS submitted via Moodle

Distributed Algorithms and Systems 4 (DAS4)
Assessed Exercise: 2013-2014

Date issued	*Friday, 25 October 2013*
Final submission deadline	*Friday, 29 November 2013*

This is the main exercise for the DAS4 module; it is worth 20% of the assessment for the course (the remaining 80% is for the May examination).

You are asked to *design* and *implement* a simple online auctioning system using Java RMI. The auctioning system should consist of an auctioning server and client programs, as described below:

The client program(s) should enable a user to create auction items (specifying name, minimum item value and closing date/time; returning unique id), and to bid against existing auction items. Methods for listing available auction items should also be included. By the end of an auction, the owner as well as the bidders of the item should be notified of the result (e.g., winner id, winning bid, price not met, etc.).

The auctioning server should be able to handle a dynamically-changing set of auctions, deal with requests from clients, and maintain the state of the various ongoing auctions. Multiple clients are expected to be launched from machines in the network and invoke the appropriate methods on the server. Auction information must be retained and queryable by a client application for a given time period after the auction closes (in reality, this would be a number of days; for the purposes of testing your program, you will need to choose an appropriate/configurable time unit). The server should also be able to save/restore auction state to/from permanent storage. The latter feature should be used for bootstrapping the server with some initial auction items (for system testing purposes).

Your auctioning system should deal with issues arising from concurrent client access to auction items, as well as from the implicit requirement to keep state for clients and auctions. Note that you are not expected to provide a sophisticated user interface for any part of the system (a simple text based interface will suffice).

You are asked to *write a report* (of no more than 3000 words) on your auction system discussing your overall design, and how your implementation meets the design goals and requirements. You should make clear (a) which parts of the specification you have actually implemented, (b) whether your solution works as advertised, and (c) how you have tested it.

The report should include a discussion of potential extensions that could improve the fault tolerance and availability of your system (you are not expected to implement these extensions!). Also, a section should be devoted to discussing the *performance* of your implementation after developing the appropriate test interfaces that will enable you to measure, e.g., max throughput of your system when responding to remote/local invocations, etc. Finally, you should include a section describing how you would expect me to test your system, given the classes, test programs, test data, etc. that you have submitted.

Warning: This is a solo exercise and must not be discussed with your classmates or anyone else. However, you can discuss the non-assessed code presented in class and any examples of Java RMI programming that you find on the Internet. You are also free to re-use code from your previous (warmup) exercises or code that has been presented in class.

What to Submit

You should submit your solution electronically via the submission link on the DAS4 Moodle page: http://moodle2.gla.ac.uk/mod/assign/view.php?id=71584

Submissions should consist of two files:

1. A single container file (e.g., a .tar, .zip, etc.) with your source code listings, and instructions on how to extract, compile, test your code.
2. A single .PDF file of your report

Both filenames should be called NNNNNNN.suffix, where NNNNNNN is your matric number, suffixed by the appropriate file extension.

IMPORTANT: Do not assume that I will be using particular IDEs/toolsets to test your code – I will not! You must therefore test your code on the School/lab Linux machines; excuses such as "it worked at home" are not acceptable. Be explicit about any issues relating to ClassPaths etc.

END/