The University of Leeds
School of Computing

TC30 (COMP3900) Coursework 2 - 2011/2012

Deadline: Friday 28 October 2011, 9:00

Introduction

This exercise aims to give you some practical experience of using Java RMI. You should review the notes on this topic (Lectures 4-7) carefully before you start work. You are provided with the bulk of the code for a distributed *City Server* system. The client of the city information server will provide a city name and get various bits of information about that city (country, minimum temperature, maximum temperature).

The source code supplied to you is not complete; there is a number of files that need completion, and a number of other changes are necessary in order to compile the system. The work required is broken down into a series of small steps that you should follow, in the order shown, to complete the exercise.

Useful Resources

Useful resources are available on TC30 module area on the VLE: `https://vlebb.leeds.ac.uk`

- Socket programming
- Database connectivity using JDBC
- Distributed applications using RMI
- Web-based distributed applications using servlets
- Web-based distributed applications using JSP.

See also Sun's (now Oracle) RMI tutorial on `http://java.sun.com/docs/books/tutorial/rmi/`.

The Task

1 Create a directory for your TC30 work. Copy the file `cwk2.tar.gz` available on the module area on the VLE in the directory *Coursework* to it. Untar this file (`tar xvfz cwk2.tar.gz`), this will create a subdirectory `ex1`, containing the source code for the city information server system. Examine this code and try to get an overall feel for what the software is doing (or supposed to do). You should focus on the RMI aspects.

2 Look at the file `CityImpl.java`, the implementation of the remote interface specified in `City.java`. You should be able to see that the implementation is incomplete.

Question 1 Complete the implementation of `CityImpl.java`. Compile the program.

3 Now focus your attention on the file `CityServer.java`. Think about the communication going on between this server and its client, where the remote objects are, and what files are needed to support the use of remote objects.

Question 2 Examine `CityServer.java`. Follow the logic inside the `main` method and explain what the server program does. Compile the program.

Question 3 Write a client program that updates a City object. This client program should invoke one or more of the methods provided by the *City* interface. Explain what you have done and why. Note at this stage that the city's details are passed as *command line arguments*, e.g. `Rome Italy 18 31`

Question 4 This is similar to Question 3. Write a second client program that displays the city's relevant information.

4 When you have finished writing the client applications code, compile the existing files. Run the server and the client on two separate terminal windows and do some tests.

Question 5 Reflect on the benefits of using RMI to implement a City server. How does this system compare with a socket-based implementation? What advantages does this system have over that implementation?

5 The next task is to have the City server system query the city database in coursework 1. To do so, you need to re-inspect the file `QueryDB.java` which you have already developed.

Question 6 Modify the City server system so that it should accept the name of a city and query the `tc30_cw1` database on `csdbdev` for the name associated with this city (instead of command line arguments, see question 3). Explain the changes that were necessary in your code.

6 The client cannot start up the server and create instances of it (this is essentially due to the server implementing `UnicastRemoteInterface`). To get around this issue, we need to create a *dynamic* server where clients should be able to start up the server and instantiate objects. A dynamic server can be created by implementing a *factory*. With the factory, you will be able to create multiple instances of remote objects and have them all interact together. Therefore, a factory would create objects for you and manage them. A factory is really a server just like the one implemented in the previous section; the only difference is that a factory would have just one method responsible for creating objects that will be later used by clients as normal servers.

Question 7 Define a factory interface whose implementation would be capable of creating a server that dynamically instantiates objects. Modify the implementation server so that given a city name, the factory will create an object for that city managed by the information server. Run the whole newly modified city information server and check whether the client displays the correct results.

7 This last part of the exercise should be attempted by those who complete Steps 1–6 quickly. The final step is to implement a class that provides an appropriate user interface for the collection of input data and presentation of results. The application should be implemented as a servlet or using JSP, as appropriate. If you are following either option, you will need to package your application in the conventional way, as a WAR file.

Submission

Submit your answers on paper in the normal way, via the CSO postbox. Remember to attach a completed coursework header sheet. To submit your code for the exercise, create a Zip or tar archive of the files which make up your system and submit the file using SIS. Remember to include a descriptive `README` file as well as any additional material.

Be sure to include all files needed for your city server system to work as part of your submission. You may include `.class` files if you wish, but this is not necessary; we will be testing whether your code compiles when we mark it.

Distribution of Marks

Question 1	3
Question 2	3
Question 3	3
Question 4	3
Question 5	4
Question 6	3
Question 7	5
Style/comments	2
	26
Servlets/JSP coding	4
	30