

## COMP30220 Distributed Systems Practical

### Lab 2: RMI-based Distribution

Work individually. Submit your code on [cs.moodle.ucd.ie](https://cs.moodle.ucd.ie) by the deadline given on moodle. Please submit a single ZIP file (name is <student-num>.zip) containing your source code and any text files required for the solution.

Please read AND run the QuoCo scenario before attempting this practical.

The broad objective of this practical is to adapt the code provided to use RMI for interaction between each of the 3 quotation services and the broker and between the broker and the client. In the final version, each of these components should be deployable as a separate docker image and you should provide a docker-compose file that can be used to deploy the images.

To help you complete this challenge, I have broken the problem up into a set of tasks. My advice is to create a separate project for your solution, and for you to copy code from the original project as needed.

#### Task 1: Setting up the Project Structure

Grade: E

As indicated above, we will break the original project into a set of projects: one of each of the distributed objects, and a core project that contains the code that is common across the overall application. Based on this, you should create a set of folders as follows:

- **core**: contains the common code (distributed object interfaces, abstract base classes & any data classes)
- **auldfellas**: The Auldfella's Quotation Service
- **dodgydrivers**: The Dodgy Drivers Quotation Service
- **girlpower**: The Girl Power Quotation Service
- **broker**: The broker service
- **client**: The client service

#### Task 2: Creating the Core project

Grade: D

The following steps will help you to transfer the existing code to the correct projects

- a) For the project, create a "src/main/java" folder and copy the "service.core" package into it.
- b) Modify the `BrokerService` & `QuotationService` interfaces to extend `java.rmi.Remote` and for the method signatures to throw `java.rmi.RemoteException` (this is the same as we did when we created the `Calculator` interface in class).
- c) Make the `Quotation` and `ClientInfo` data classes implement the `java.io.Serializable` interface.
- d) Copy the `pom.xml` file from the "core" project of the RMI Calculator. Change the **groupId** to "quoco" and the **artifactId** to "core".
- e) Compile & Install the "core" project

### Task 3: Creating and Testing the Distributed Quotation Services

Grade: C

The second task involves creating a distributed version of the Quotation Services. I will start by explaining how to do it for one of the services – auldfellas – and you will need to do the same thing for the other services.

- a) Copy the RMI Calculator client **pom.xml** file into the auldfellas folder. Modify the groupId to be “quoco” and the artifactId to be “auldfellas”. Set the main class to be a new java class called “Server” (we will create this shortly). Remember to update the dependency to reference the “core” project.
- b) Create the “src/main/java” folder structure and copy the auldfellas package into it.
- c) Create a class in the default package called `Server.java` - this is the one we mentioned in part (a). Copy the code below into it.

```
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;

import service.auldfellas.AFQService;
import service.core.ClientInfo;
import service.core.Quotation;
import service.core.QuotationService;
import service.core.Constants;

public class Server {
    public static void main(String args[]) {
        QuotationService afqService = new AFQService();
        try {
            // Connect to the RMI Registry - creating the registry will be the
            // responsibility of the broker.
            Registry registry = LocateRegistry.createRegistry(1099);

            // Create the Remote Object
            QuotationService quotationService = (QuotationService)
                UnicastRemoteObject.exportObject(afqService, 0);

            // Register the object with the RMI Registry
            registry.bind(Constants.AULD_FELLAS_SERVICE, quotationService);

            System.out.println("STOPPING SERVER SHUTDOWN");
            while (true) {Thread.sleep(1000); }
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }
}
```

Try to understand what this class is doing – it is basically the same as the CalculatorServer class, but that it is creating the Auldfellas Quotation Service distributed object.

Try to compile & run the “auldfellas” project.

- d) The next step is to write some code to test this service. To do this, we will do some work on the “client” project. This will be a temporary version of the client code that we will use to test the quotation service we have just created.

Copy the pom.xml file you created in step (a) of this task into the client folder. Modify the **artifactId** to be “client” and the main class to be “Client”. Generate the “src/main/java” folder structure and create a file called Client.java.

Copy the displayProfile(...), displayQuotation(...) and clients data from the client.Main class in the original project. Use the following main() method to test the service:

```
public static void main(String[] args) {
    String host = "localhost";
    if (args.length > 0) {
        host = args[0];
    }

    QuotationService quotationService = null;
    try {
        Registry registry = LocateRegistry.getRegistry(host, 1099);

        quotationService
            = (QuotationService) registry.lookup(Constants.AULD_FELLAS_SERVICE);

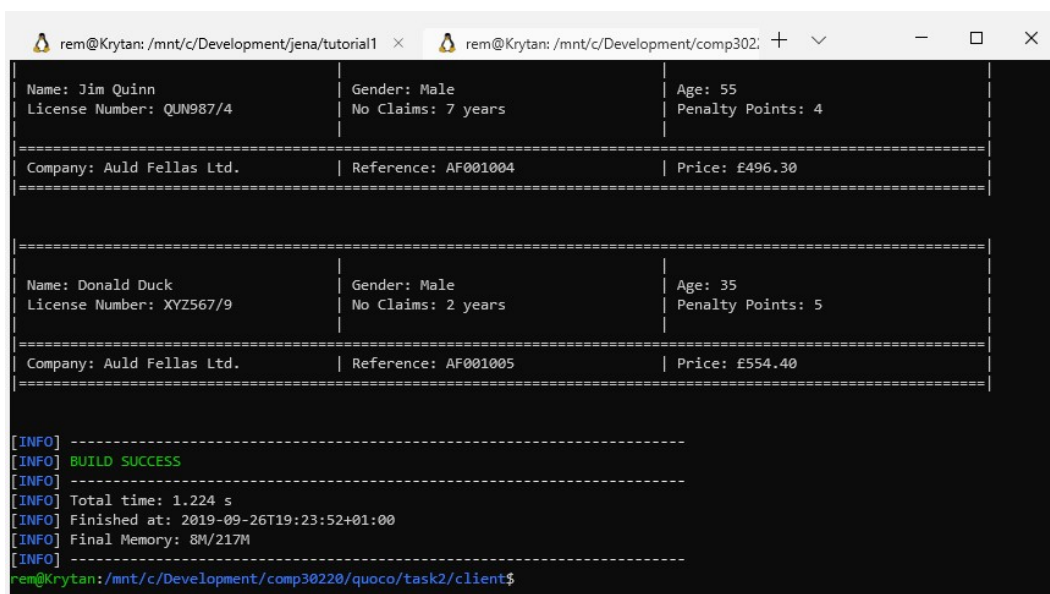
        for (ClientInfo info : clients) {
            displayProfile(info);

            Quotation quotation = quotationService.generateQuotation(info);
            displayQuotation(quotation);

            System.out.println("\n");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Notice that this code is similar to the client code used in the original version of QuoCo with the exception that only a single service is contacted and only a single quotation returned (the broker returned a list of quotations).

Compile and run this code -you should get something like the screenshot below:



```
rem@Krytan: /mnt/c/Development/jena/tutorial1 x rem@Krytan: /mnt/c/Development/comp302: + - □ ×

Name: Jim Quinn      Gender: Male      Age: 55
License Number: QUN987/4  No Claims: 7 years  Penalty Points: 4
=====
Company: Auld Fellas Ltd.  Reference: AF001004  Price: £496.30
=====

Name: Donald Duck    Gender: Male      Age: 35
License Number: XYZ567/9  No Claims: 2 years  Penalty Points: 5
=====
Company: Auld Fellas Ltd.  Reference: AF001005  Price: £554.40
=====

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.224 s
[INFO] Finished at: 2019-09-26T19:23:52+01:00
[INFO] Final Memory: 8M/217M
[INFO] -----
rem@Krytan: /mnt/c/Development/comp30220/quoco/task2/client$
```

- e) Once you are convinced that service works, you will need to make a last change to the Server class so that it does not try to create the RMI Registry – this will be created by the broker in the final system.

Add the code that takes a hostname as a parameter:

```
String host = "localhost";
if (args.length > 0) {
    host = args[0];
}
```

Replace:

```
Registry registry = LocateRegistry.createRegistry(1099);
```

With:

```
Registry registry = LocateRegistry.getRegistry(host, 1099);
```

- f) Repeat the above to create and test the “girlpower” and “dodgydrivers” projects.

#### Task 4: Implementing the Broker and Client

Grade: B

Now we have working quotation services, the next task is to create and test the broker.

- a) Expose the LocalBrokerService as a distributed object and modify the local broker service to use the RMI Registry to find the quotation services. This should all be implemented in the “broker” project.
- b) Modify the test client to lookup the broker and modify the main() method to loop through and print out all the quotations returned by the broker service.
- c) Compile and run both projects 😊

#### Task 5: Containerisation

Grade: A

The final task is to convert the output of task 3 into a set of docker images that can be run from a docker compose file.

#### Additional Marks

+ grades (e.g. A+) can be attained through consideration of boundary cases, good exception handling, nice features that enhance the quality of your solution.

- grades (e.g. A-) can be attained through lack of commenting and indentation, bad naming conventions or sloppy code.