Student Name: **Id:**

Laboratory Goals / Objectives

Upon completion of the lab, the individual should be capable of developing a simple client / server type application using Remote Method Invocation (RMI).

Task 0

Copy the following code sample and paste it into an editor such as TextPad naming the files appropriately. Read through the programs to see how the *sayHello()* method is defined and implemented.

File Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

     public Server() {}
     public String sayHello() {
          return "Hello, world!";
     }

     public static void main(String args[]) {
          try {
               Server obj = new Server();
               Hello stub =
          (Hello)UnicastRemoteObject.exportObject(obj, 0);
          // Bind the remote object's stub in the registry
               Registry registry =
               LocateRegistry.getRegistry();
               registry.bind("Hello", stub);
               System.err.println("Server ready"); }
          catch (Exception e) {
               System.err.println("Server exception: " +
          e.toString());
               e.printStackTrace();
          }
     }
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
     private Client() {}
     public static void main(String[] args) {
     String host = (args.length < 1) ? null : args[0];

     try {
          System.out.println("Host: " + host);
          Registry registry =
     LocateRegistry.getRegistry(host);
          Hello stub = (Hello) registry.lookup("Hello");
          String response = stub.sayHello();
          System.out.println("response: " + response); }
     catch (Exception e) {
          System.err.println("Client exception: " +
     e.toString());
          e.printStackTrace();
     }
     }
}
```

File Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
     String sayHello() throws RemoteException;
}
```

Task 1

To compile the above "hello world" application one can use the standard: javac *.java to compile all the files. Before the applications can be run, the rmiregistry must be running so that remote objects may be registered. Open a prompt and enter one of the following:
rmiregistry (to run within that window), start rmiregistry (to run within a new window)

To run the Server: start java Server

To run the Client: java Client

Task 2

Rename the Hello interface to something more suitable and include additional methods for mathematics. The interface should include methods to return PI, add(num1, num2), divide, multiply, subtract. A simple GUI client application should be developed using either SWING or AWT components, to provide a calculator front end. Two text boxes for input, and perhaps another one or a Label to display the result. The GUI should also have a Combobox or radio buttons etc, to allow the user to select the type of operation to perform (add, divide etc.). All the processing should be carried out on the server when the user presses the "Calculate" button.

Task 3

Having successfully completed the Calculator one should now add additional functionality to the Server. The Server should maintain a small database of records. The particulars of the "database" and the fields it contains are down to individual preference. Some possible ideas: Car Sales Database, Time Table Database, Cinema Listings Database, etc. A variety of datatypes should be used, for example: String, int, float, double, boolean.

Note: The database should be initialised with a few records. One may use an Array, a Vector or some other type of object to act as the database on the Server.

The Server should have an additional remote object (One new interface and one implementation file, therefore a total of 2 additional java files need to be created). The remote object should provide a set of "Admin" methods to Add additional records to the database, list all records, search, and remove records. It should also provide methods for "Ordinary users" allowing them to list a limited subset of the database, and allow for modifications to the database. In the case of a Car database, an ordinary user should be able to get a listing of the Cars primary key, the make, and the price. A facility should then be available to allow for the purchase of a Car.

Two additional Client applications should be developed: one to manage the "Administration" of the database, the other for "Ordinary Users". The Clients may be text-based running from a command prompt or, have a simple GUI.

Note: If you make a change to the Server and wish to run it again, you will need to terminate the registry before running the Server again.

Questions

The following questions are to be filled in individually by each student.

1. What is RMI?

2. What is the default port on which the rmiregistry runs? Provide an example of how to start the rmiregistry running on a different port?

3. What is the purpose of the rmiregistry? What exactly does it do?

4. Provide the interface definition for the calculator

5. Provide the Server implementation for the calculator

6. Provide the interface for the database application

7. Provide the implementation of the remote object for the database application

8. Provide the code showing the registration process of all the remote objects running on the Server

9. Provide the code for the Admin Client

10. Provide the code for the Ordinary User Client

11. Provide the screenshots of your applications' executions.