Practical 3

Theme

- Accessing objects on a remote computer using Java RMI
  - communicating with a RMI registry
  - executing remote methods
  - passing a serialisable object as a parameter
  - passing a remote object as a parameter

Key concepts: remote object, remote method invocation, registry, remote reference, serialisation

3.1. Start up and essential configuration

a) *Log-in to Ubuntu, start terminal.*

b) *Download and extract source code archive* `lab3-client.zip` *from BlackBoard.*
Extract it into your `CS3250-DS-1011` folder.

> **Important.** It is essential that you place all your Java and Eclipse related files into your `CS3250-DS-1011` folder that you created as a symbolic link (a form of folder shortcut or junction) in Practical 1. This folder appears in your H: drive but is physically on another drive, which is available only in Ubuntu. This arrangement is in place because the `H:` drive does not work reliably on Ubuntu in the labs.

c) *Get familiar with the concept of multiple desktops.*
By default your Ubuntu account has several virtual desktops (usually 4). You can switch between them using the `Ctlr-Alt-Left/Right/Up/Down` keys. By opening eg the browser in one desktop, terminal in another and Eclipse in yet another you may find it easier to switch between applications.

d) *Open the code in eclipse 3.6 with new workspace in* `CS3250-DS-1011/workspace`*.*
To start Eclipse 3.6, either press `Alt-F2` and enter `eeclipse.sh` or type the command `eeclipse.sh &` to a terminal.
Create a new workspace in folder `CS3250-DS-1011/workspace`, then switch the perspective from JavaEE to Java and select File > New > Java Project. The New Java Project dialogue box will open. Select the Create project from existing source radio button and click browse to navigate to the location of the extracted source code eg. `CS3250-DS-1011/lab3-client`. Click Finish.

e) ***(Only if you work outside the lab session)***
Download, extract and execute `lab3-server.zip`*.*
In terminal, `cd` to the extracted `lab3-server` folder and run:

```
javac */*.java
sh runServer.bat
```

A new window should appear with several coloured rectangles.
To stop the server, press Ctrl-C at the terminal where it was started.
Important: If you run your own server, in subsequent tasks always replace `duck` with `localhost`.

3.2. Obtaining a list of remote server objects

a) *Complete the missing code in `clients/LookupServers.java` to locate and interrogate the server's registry.*
Complete the code in the places marked 'TODO'.
Hints: Use the `java.rmi.registry.LocateRegistry` class to obtain a `Registry` object linking to the registry of remote objects on the remote server. The host name of the server is `duck`. You should assume that the default port is being used (ie 1099).
Use the `list()` method of the `Registry` class to obtain an array of remote objects registered with the RMI registry. The names obtained will be used later in this lab class.

b) *Run the program and observe the response from the server.*
Hint: To run the program on command line, *`cd`* to the `lab3-client` folder and execute the command *`sh Q2.bat`*.
Hint: To run the program via Eclipse, you need to use the Run Dialogue, select the `LookupServers` Java application, in the Arguments tab enter the following into the VM arguments entry box:

```
-Djava.security.policy=policy.all
```

3.3. Simple interaction with the server

a) *Examine the file `server/BoardServerInterface.java`.*
This file shows what remote methods can be called from our programs. In this part we will use the `sayHello` method.

b) *Complete the file `clients/SimpleHello.java`.*
Complete the code wherever there is a 'TODO'.
First, use the `java.rmi.Naming` class to look up the remote server object using the name that was obtained in part 3.2.
Once a reference to the remote object has been found, call its `sayHello` method. Note that it requires a single `String` parameter and returns a `String`.

c) *Run the program and observe the response from the server.*

3.4. Lines and shapes: passing object parameters

The aim of the following exercises is to draw lines and shapes. The drawings will be displayed in the server's display on the large screens. Notice that lines are displayed on one canvas and shapes on another canvas. Both canvases measure 300×300 pixels.

a) *Examine the file `server/BoardServerInterface.java` again.*
Recall that this file shows what remote methods can be called from our programs.
In this part we will use the `drawLine` and `drawShape` methods. The `drawLine` method requires a parameter of type `Line` defined in the file `objects/Line.java`. Similarly, `drawShape` requires a parameter of type `Shape` or its extensions available in the `objects` package.

b) *Complete the file* `clients/DrawLetter.java` *so that a line and/or shape is drawn on the server.*

Use the `java.rmi.Naming` class to look up the remote object that was obtained in part 3.2. This code will be identical to that used in task 3.3.

Once a reference to the remote object has been found, code calls to its `drawLine` or `drawShape` method with appropriate parameters.

c) *Execute the program and observe its effect on the server screen.*

You can do this repeatedly for different shapes.

d) *Attempt to draw a letter of the alphabet.*

e) *Explore what happens when a parameter class is modified on the client but not on the server.*

This exercise assumes your program works fine and uses the class `Line`.

Alter the `serialVersionUID` constant in file `objects/Line.java`. (It does not matter how you change it; perhaps modify one of its decimal digits.)

When Eclipse has compiled the program, execute it again and observe its effect. The program should fail saying that the server does not recognise the `drawLine` method. The reason for this is that the class `Line` on server is not the same as the class `Line` on the client and thus the method `drawLine` appears to have a different parameter type.

3.5. Lost Duck? Passing remote objects as parameters

Dr. Tribbiani has lost his pet duck. He has asked 4 students to look for it. The position of the duck and the 4 students can be seen in green area at the bottom section of the large screen. Each student is denoted by a number.

The aim of the following exercises is to move each student (ie player) around the playing field to the location where the yellow duck can be found. The field is a grid of 30×15 squares.

a) *Examine the file* `server/BoardServerInterface.java` *again.*

In this part we will use the `movePlayer` method on line 56 (beware, there is another `movePlayer` method nearer the end of the file), which moves a specified student player one square in a specified direction. Two parameters are required: an integer to indicate which student to move (1,2,3,4) and a member of the `Direction` enumeration type, which is defined in file `objects/Direction.java`.

b) *Complete the file* `clients/SearchDuck.java` *to instruct one of the 4 students to move.*

First of all, repeat the modifications that you made in the previous exercises in order to obtain a reference to the remote server object.

Using the server's `movePlayer` method, try and implement a simple algorithm to try and find the duck. Unfortunately, once a player moves to the duck's location, it will fly away to another location.

c) *Execute the program.*

Run the program and observe the response on the large screens. You may see that the players move chaotically as you and your fellow students try to move the same player all at once.

d) *Examine the interface* `server/BoardServerInterface.java` *yet again.*

This time pay attention to methods `createPlayer`, `getPlayer` and `movePlayer` on lines 59–93.

e) *Modify your* `SearchDuck` *program so that it creates its own, suitably named, player instead of using one of the pre-existing four student players.*

Notice that you need to work with an interface (ie `PlayerInterface`) rather than a class. This is because your player resides on the server — it is a remote object.

Hint: you can use the player's `getXPos()` and `getYPos()` methods to find out where in the grid it is.

3.6. (Optional) Explore and modify the server code.

a) *Get and deploy locally the server code.*

Use instructions from 3.1.(e).

b) *Compare the files* `objects/PlayerInterface.java` *on the server and on the client.*

c) *Explore the definition of the method* `movePlayer(PlayerInterface, Direction)` *in files* `BoardServer.java` *and* `pnlPeople.java`*.*

d) *Add a third parameter to the method* `movePlayer` *to indicate how long a step the player should make in the given direction.*

e) *Test the method using a modified* `SearchDuck` *client.*