COMP28112 Exercise 1: Servers and Clients
Duration: 1 session (deadline: end of the lab)

Learning outcomes

On successful completion of this exercise, a student will:-

- Have run and modified a server written in Java using sockets
- Have run a client written in Java using RMI
- Have run and modified a server written in Java using RMI
- Have run and modified a Java servlet

Introduction

Distributed systems often make use of the client-server model of interaction, and there are many different ways of implementing clients and servers. This exercise will illustrate three of the more simple mechanisms for this, namely: sockets (in Java), Remote Method Invocation (in Java), and Java servlets. For each you will be supplied with working code which implements a very simple interaction. You should run this, to demonstrate to yourself how the interaction is achieved, and then modify it to perform a slightly more interesting interaction. In all cases the client and server can be run on the same machine; it should, however, also be possible to interract with other machines when they are running an appropriate server unless security constraints prevent this.

Initial behaviour

The initial behaviour of each of the clients is to send a message, just a string, to the server. The initial behaviour of each server is to report the strings received from clients, and to send each a fixed string in reply. The clients will report this result.

Target behaviour

The string returned by each server should be one chosen as a name by the person running it, but only if the correct string, “whoRU”, is sent by the client. Any other input to the server should generate the message “If you ask me nicely, I will tell you who I am”.

Sockets

`$COMP28112/ex1/SocketServer.java` [1] contains the initial code of a server using sockets. The server waits for requests on port 8181, and starts a separate thread to process each one received. If you compile and run this it will listen for requests from clients (on port 8181). To generate such a request, use telnet - as in:

```
telnet ug999 8181
```

where `ug999` is the name of the machine on which the server is running. When the connection has been made, standard input is then sent to the server. Type what you want to send and it will be transmitted when you press the newline. telnet will write the reply it receives to standard output. The easiest way to do this on a single machine is to run the server and telnet in separate shells. When you modify the server, you should make the server's identification response derive from the command line used to run it.

RMI

Remote Method invocation in Java is a mechanism whereby client code can invoke (i.e. call) a method which runs on the server. `$COMP28112/ex1/RMServer.java` contains the initial code of such a server, and `$COMP28112/ex1/RMClient.java` contains the client code. To compile either of these classes requires the presence of `$COMP28112/ex1/RemoteServer.java`, an interface which identifies the method which the client can call in the server.

Compile and run these; as before it will probably be simpler to use two separate shells. Note that the server needs the RMI Registry to be running when the server process is run. The server process registers itself with the Registry, which then redirects requests from the client to the server process. If it is not already running on the server machine, use the command:

```
rmiregistry &
```

to run it as a background process before starting the server process (NB: you have to make sure that the directory where `rmiregistry` is located is in your path; if it is not, you may need to do some research or use `which` to locate it). Now modify the server as required; again the identification string should be derived from the command line used to run the server (in other words, you should make the 'name' of your server an argument of the server program). The client should need no modification. Note that versions of Java earlier than 1.5 required an extra step, using `rmic` in the server, but this is no longer necessary.

Servlets

Servlets are Java's way of providing CGI, and the client, therefore, is normally a web browser. A servlet is a Java program which runs in the server, e.g. under Apache Tomcat, and the `doPost()` method is invoked for each http request posted to it. The servlet responds to the client by generating html which the browser then displays. To debug a servlet, instead of giving it to Tomcat, you can run it using the command:

`$COMP28112/ex1/servletrunner -d ~/COMP28112/ex1` or

`$COMP28112/ex1/servletrunner -d /home/<your_username>/COMP28112/ex1`

where the final item is the absolute pathname of the directory containing the classfile of the

[1] In case `$COMP28112` is not properly set up, please note that this points to `/opt/info/courses/COMP28112/`

servlet. That directory should also contain a file called `servlet.properties` which identifies the code to be used for a particular service, and can also supply some initial arguments for it. The latter are needed because servletrunner, and any other way of hosting servlets, can support a number of different servlets and they are only initialised when needed. It does not make sense, therefore, to provide initial arguments on the command line.
`$COMP28112/ex1/servlet.properties` illustrates the format used, and the servlet provided echoes an initialisation parameter to show how its value is obtained. The initial code of a servlet is provided in `$COMP28112/ex1/SimpleServlet.java`. To compile it you will need to type:

```
javac -classpath ./jsdk.jar SimpleServlet.java
```

(NB: you should copy `$COMP28112/ex1/jsdk.jar` into your `ex1` folder)

The html for the client is in `$COMP28112/ex1/ServletClient.html`. It is common for such clients to have the name of the server machine within the html, but here some simple JavaScript is used to provide more flexibility. When you change the code in SimpleServlet.java, make it take the identity information from the `servlet.properties` file.

Assessment

Marks will be awarded as follows:

Correct working and explanation of new socket server	2
Correct working and explanation of new RMI server	2
Correct working and explanation of new servlet server	2

(note that to get full marks you will have to show that you understand what you are doing)

Do this exercise in your `COMP28112/ex1` directory. `labprint` will look there for:

```
SocketServer.java
RMServer.java
```

and `SimpleServlet.java`.

Remember to `submit` your solution.