UNIVERSITY of STIRLING

CSCU9V5: Concurrent & Distributed Systems

Distributed Laboratory 1

This lab contains two checkpoints.

The server-client code presented in the lecture is provided as a starting point. The initial task is to run a server and a client. The client accesses the time from the server. Initially the server and client run on a single host.

Launch server and client, and check that all works as expected. This will also be useful to set up your working environment. For instance, in eclipse, you should be able to have one console window for each client or server. Each console window should be detached, i.e. outside the main eclipse window.

Try to launch the server twice. This will cause the second one to exit with a runtime exception.

Q1 Explain why and point the line in the server code where the exception is generated.

Add a suitable loop to the server and one to the client so that the both can deal with an "infinite" number of requests generated by the client and served by the server. Add suitable messages to be printed showing the interaction. Please add counters for requests and responses. Also print the name of the active thread as appropriate in the relevant messages (check which class has threads).
You may find `java.lang.Thread.getName()` useful with `System.out.println(..)` in the server-connection code to follow what is happening.

Try running more than one client at the same time (you need one console window per client and server). Check that all works as expected.

Stop the server (by means of the button on the console window). The client(s) exit with a runtime exception.

Q2 Explain why and point the line in the server code where the exception is generated.

Next, have the server and client on separate hosts. Again, have more than one client operating at once. You may find `java.net.InetAddress.getLocalHost()` useful with `System.out.println (..)` in both the server and client to follow what is happening.

This might be along the lines of :

```
// as an experiment let's get the IP address of the server
   try{
      InetAddress server_inet_address =  InetAddress.getLocalHost() ;
      String server_host_name = server_inet_address.getHostName();
      System.out.println ("Server hostname is " +server_host_name ) ;
      System.out.println ("Server port is xxxx") ;
   }  // end of try
   catch (java.net.UnknownHostException e){
      System.out.println(e);
      System.exit(1);
   }  // end of catch
```

Also, it will be easier if you modify the client along the lines of the following code. This will allow you enter the server name and port number when you run your client. :

```
public static void main(String argv[]) {
  if ((argv.length < 1) || (argv.length > 2))
      { System.out.println("Usage: [host] <port>") ;
        System.exit (1) ;
      }

  String server_host = argv[0] ;
  int server_port = 5156 ;
  if (argv.length == 2)
      server_port = Integer.parseInt (argv[1]) ;

  Client client = new Client(server_host, server_port);
  } // end of main
```

Try your client with someone else's server.

Checkpoint: Show answers to **Q1** and **Q2,** two clients and a server on the same computer printing suitable messages**.**

Now, the server changes and returns some `double` values every time that a client connects, The client connects to the server, gets the values, and prints their sum. To carry out such arithmetic operations it is better not to send the information as strings. So instead of writing out text with the PrintWriter class, write the data as encoded data with the DataOutputStream class. So:

```
pout = new PrintWriter(outputLine.getOutputStream(), true);
```

becomes

```
poutdata = new DataOutputStream(outputLine.getOutputStream());
```

The `DataOutputStream` supports methods such as `writeDouble`, as in

```
try {poutdata.writeDouble ( 1.123456789012345 );}
catch (java.io.IOException e)
  { System.out.println(e);
    System.exit (8) ;
  }
```

Note you need to handle exceptions. You *can* output doubles as text with

```
pout.println(1.123456012345) ;
```

But the client can only receive it as a `string`. You might convert the `string`, but it's better to use the `readDouble` method of the `DataInputStream` object. The client can receive the data simply by creating a `DataInputStream` object:

```
ip = new DataInputStream(in);
```

And using its `readDouble` method to access and decode the data. Note you now have the received the data as a double which you can add etc. to other variables:

```
double local_double = 0.0 ;
local_double = ip.readDouble() ;
```

Checkpoint.

Note: At the end of the day its all transmitted as bytes - its simply a matter of how the data is encoded and subsquently decoded. Here both server *and* client know the data structure and size and are programmed appropriately.

The java API documentation helps.