COMP30220 Distributed Systems Practical

Lab 1: Simple Sockets

Work individually. Submit your code on csmoodle.ucd.ie at the end of the session.

1. Download the socket sample code from the Moodle.

2. Read through the client and server code and make sure you understand it.
   Try to work out what the code does purely from inspection and the last lecture.

3. Run the server code then the client code. For example:
   ```
   java TCPSocketServer 1234
   java TCPSocketClient localhost 1234 "hello world"
   java TCPSocketClient 127.0.0.1 1234 "hello world"
   ```

   Does the code do what you expected? Ask a demonstrator if anything is unclear.

   Note: you can run the example code from a single machine (e. g. from different command shells) or try running the server code on one machine, and the client code on another. In the latter case, make sure both machines are connected to the same access point; otherwise you may have problems connecting to certain port ranges.

   Read ALL the following points before starting to solve them, <u>spend time designing your system before implementing it</u>: you will get 40% of the overall grade for this exercise if you correctly solve point 4, 70% if you solve 4 and 5, 90% if you solve 4, 5, and also provide an efficient solution to 6.

4. Alter the code so that the server sends back to the client "`Your message was: <message>`". The client should print this response out to the console.

5. Change the code to implement a "division service", i. e. the client will send two numbers and the server will need to respond with the result of dividing the first number with the second.
   Example (testing on the same machine on port 1234):
   ```
   java TCPSocketClient 127.0.0.1 1234 80 3
   ```
   The client should output the following: `"The result of 80/3 is 26.66"`

6. Change the client and the server code to handle 100 random division requests (the client extracts two random numbers and sends them to the server, which divides them before sending the result).

   Make the communication more efficient, by:
   - creating the client socket only once
   - use `DataOutputStream` and `DataInputStream` to transmit binary encodings of the numbers and the result.

   Extra points will be awarded depending on the overall quality (e. g. modular design separating networking code from functional aspects, correct handling of exceptions...) Points will be taken for mistakes impacting the efficiency and the modularity of the design.