THE UNIVERSITY OF SYDNEY

COMP2121 Lab 3

Client-Server Communication

The goal is to write a client-server communication using sockets. Your machine will be used both as a client and as a server, and the client will simply ask the time of the server sent back to him in **`String`** format. Finally, the client should write the information received on the standard output.

Exercise 1: "What time is it?" server

The server, written in a class **`DateServer`**, should always be listening to some incoming connection on a non-reserved port, e.g., 6013, and treating it. The server needs an **`OutputStream`** to send information in response to some client request as described below. Although not necessary here, you are generally welcome to check the Java API (http://docs.oracle.com/javase/7/docs/api/) to learn more about classes you may need that already exist, like **`OutputStream`**. Complete the code below using **`Socket`** in order to accept an incoming connection and write it in file **`DateServer.java`**.

```
1  import java.net.*;
2  import java.io.*;
3
4  public class DateServer {
5      public static void main(String[] args) {
6          try {
7              ...  // TODO: create a new socket
8
9              // now listen for connections
10             while (true) {
11                 ...  // TODO: wait and accept a client connection
12                 PrintWriter pout = new PrintWriter(client.getOutputStream(), true);
13
14                 // write the Date to the socket
15                 pout.println(new java.util.Date().toString());
16                 ... // TODO: close the client connection
17             }
18         } catch (IOException ioe) {
19             System.err.println(ioe);
20         }
21     }
```

Duration: 15 min

Exercise 2: "What time is it?" client

The client, written in a class **`DateClient`**, simply initiates the connection by creating a socket targeting the local machine, always identified by the IP address "127.0.0.1" and the service port chosen. It reads from this connection using an **`InputStreamReader`** object and bufferizes this stream of information, before being able to print it as indicated in the incomplete code below.

```
...
// read the date from the socket
String line;
while ((line = bin.readLine()) != null)
    System.out.println(line);
...
```

Write the client class in file **`DateClient.java`**. Compile them, then run the server in the background before running the client.

```
$ javac DateClient.java DateServer.java
$ java DateServer &
$ java DateClient
```

What happens if the client runs while no server is running?

Make sure to kill the server process that runs in the background when you are done.

Duration: 15 min

Exercise 3: Logging Server

The goal is to design a new **`LoggingDateServer`** that will log each received request from a client in addition to answering the date to the client (as before). Each time the server receives a request from a client, it stores a line in a new file whose name contains a monotonically increasing number, writing for example files **`log0.txt`**, **`log1.txt`**, **`log2.txt`**, upon reception of the three first requests.

As I/O can take a non-negligible amount of time, what could be the problem in the proposed solution?

Write the code of the server so that it writes the client hostname and the sent date to the log file. Note that the client hostname can be obtained on the server side by calling **`socket.getInetAddress().getHostName();`** where **`socket`** corresponds to the instance of the **`Socket`**. Test that it creates log files, when running the previous client.

```
$ javac LoggingDateServer.java
$ java LoggingDateServer &
$ java DateClient
```

Duration: 20 min