Week 12 Lab - Practical Test: Implementing Network Communication in Java

COSC244

Assessment

This lab is worth a maximum of 6% of your total mark for Cosc 244. Each lab stream is divided into two 50-minute time slots during one of which you will sit the test. The labs are at 9am, 1pm, and 3pm on Thursday of week 12.

Please make sure you come to the lab, with your ID card, at least five minutes before the lab begins.

Overview

In the previous labs, you could complete the work at any stage before the lab finished using whatever resources you wished to. In this lab, you must complete the task during a 50-minute time slot with minimal resources at your disposal. Using only a terminal window, a blank text-editor, *javac*, and *java*, you must write a program from scratch that meets the specifications given below. This test consists of two separate parts.

1. In the first part (*worth 2%*), you must write the application which corresponds to 'Program 1' of Lab 8. This code consists of *ReadWriteThread.java*, *Server.java* and *Client.java*. This application allows the client and server to connect to one another and send messages without needing any coordination.

   We require you to make the following change to the examples covered in labs.

   Change the *ReadWriteThread* (RWT) so that it takes a String *prefix* as a third parameter to the constructor. This prefix should be added to every line which is output by the RWT. If you create a RWT with the empty string as a prefix it should behave exactly like the examples you looked at in the lab. However, if you create some of your RWTs with a prefix of `"--> "` then you will get output which matches the sample output on the last two pages of this document.

   Once you have successfully completed the first part and had it checked by a demonstrator, you can move on to the second part.

2. In the second part you can choose **one** of the following two options (we suggest you choose to implement a server that you fully understand).

   (a) (*worth 2%*) Write a more advanced server corresponding to 'Program 2' of Lab 8. This version should be called *MultiServer.java*. It allows multiple clients to connect to the server. A line typed into any client is sent to the server. A line typed into the server is sent to all clients.

   (b) (*worth 4%*) Write a more advanced server corresponding to the one in Lab 9. This version is called *ChatServer.java*. It allows multiple clients to connect to the server. A line typed into any client is sent to all clients including the originator. The server's window prints status messages but you don't type in the server's window.

You are allowed to have a maximum of two attempts at this practical test. In addition to sitting the test during your own lab stream, you may find another empty slot during your own, or another lab. If you are not confident of your ability to pass, it is recommended that you come to the early stream so that, if necessary, you will have a gap between your first and second attempt.

Some points to note

- You are not permitted to access your home directory or any other files or computers. You may NOT use the internet during this lab.
- You must successfully complete the simplest program before attempting to write one of the more complex ones.
- You must write three Java classes (*ReadWriteThread*, *Client*, and *Server*) for the first program. However you only need to write a new server class (*MultiServer* or *ChatServer*) when writing one of the more complex programs.
- In order to make marking a smooth and simple process, please write code which matches the sample code as closely as possible. Sensible variable names should be used.
- For the servers which require synchronization you must use the *synchronized()* keyword in all of the required places. You are **not** allowed to just use, for example, a *Collections.synchronizedList* instead.

If you are using Emacs, you may find it useful to use its dynamic completion feature. Type the first part of a word and then press TAB to get Emacs to cycle through each word in currently open buffers which start with what you have typed. Using this feature will considerably reduce the amount of typing you have to do.

A sample of the output which you could expect from each of the three programs is given on the next two pages. The text prefixed by `-->` represents lines which have been received from the other end of the connection.

If you have any questions or concerns about this lab and the way it will be assessed please see Iain Hewson or email *ihewson@cs.otago.ac.nz*.

Program 1 (two-way communication)

(Server)

```
Waiting for a client to connect
Accepted connection on port 7775
Hi
I'm the server
--> Hello
--> I'm the client
```

(Client)

```
Connected to localhost on port 7775
--> Hi
--> I'm the server
Hello
I'm the client
```

Program 2 (multiple connections)

(MultiServer)

```
Waiting for a client to connect
Accepted connection on port 7775
You are the first client
--> Hello
Accepted connection on port 7775
I have two clients now
--> I must be the second client
--> I have to go now
Goodbye
--> I'm still here
```

(Client 1)

```
Connected to localhost on port 7775
--> You are the first client
Hello
--> I have two clients now
I have to go now
--> Goodbye
```

(Client 1 disconnects at this point)

(Client 2)

```
Connected to localhost on port 7775
--> I have two clients now
I must be the second client
--> Goodbye
I'm still here
```

Program 3 (communication between clients)

(ChatServer)

```
ChatServer started
Accepted connection from client_1
Accepted connection from client_2
Sending 'Hello' to : [client_1, client_2]
Sending 'How are you?' to : [client_1, client_2]
Accepted connection from client_3
Sending 'I'm fine' to : [client_1, client_2, client_3]
client_1 closed connection!
Sending 'Who are you talking to?' to : [client_2, client_3]
```

(Client 1)

```
Connected to localhost on port 7775
--> Welcome! You are client_1.
Hello
--> client_1: Hello
--> client_2: How are you?
I'm fine
--> client_1: I'm fine
```

(Client 1 disconnects at this point)

(Client 2)

```
Connected to localhost on port 7775
--> Welcome! You are client_2.
--> client_1: Hello
How are you?
--> client_2: How are you?
--> client_1: I'm fine
--> client_3: Who are you talking to?
```

(Client 3)

```
Connected to localhost on port 7775
--> Welcome! You are client_3.
--> client_1: I'm fine
Who are you talking to?
--> client_3: Who are you talking to?
```
