DCU School of Electronic Engineering Assignment Submission

Student Name(s):	**Michael Lenehan**
Student Number(s):	**15410402**
Programme:	**B.Eng in Electronic and Computer Engineering**
Module Code:	**EE402**
Lecturer:	**D. Molloy**
Project Due Date:	**17/12/2018**

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited andacknowledged within the text of my work. I understand thatplagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: ***Michael Lenehan***

Assignment 2

Michael Lenehan

Introduction

The aim of this assignment is to implement a graphical server application, which will display the CPU Temperatures of the Client device in the form of a graph against time. The Client device will connect to the Server, passing the current temperature to be stored. The Server will store the last twenty input values, graphing these along with an average temperature value for all currently connected clients. The frequency at which the Client will send the temperature reading to the Server must be variable, being set via the Server GUI.

The Server application must be able to handle multiple Client connections at any time, The custom object type sent from the Client must include the following information: Temperature Data, Device Identifier, Current Date and Time, Current Sample Number.

Procedure

The completion of this assignment took place across two main steps. The first of these being the design and planning of the components required, and the second being the implementation of this design.

2.1 Design

The design of this assignment began with the design of the object class which would be sent by the Client. This object must be able to store all of the aforementioned data, including temperature, device ID, date and time, and sample number. The temperature data can be read from the "/sys/class/thermal" directory on the Raspberry Pi, or on a Linux PC for testing.

The Client class is the next class to be designed, as it will transmit the object data. The Client object must be able to connect to the server at the IP address specified by the command line arguement used to initialise the program. The object containing the

temperature information must then be sent to the Server, and a sampling frequency must be obtained from the server. The client will wait for the specified amount of time before transmitting the next object.

The next class to be designed following the Client is the Server class. The Server class must start the GUI window, and then begin to listen for Client connections. For this application, as the Server must be able to handle multiple connections at once, it must implement a connection handler that utilises multithreading. As such, when a Client connection is attained, the connection is passed to a ThreadedConnectionHandler class, which will perform all necessary operations and communications with the Client object.

The ThreadedConnectionHandler class is used to interact with the Client object once it has connected to the Server. The Server object the Client's socket information to the ThreadedConnectionHandler, which must then begins to listen for input from the Client. Once a tramsmission is received from the Client, the ThreadedConnectionHandler must store this object to an array, which will hold the last 20 values passed from the Client. The ThreadedConnectionHandler must also respond to the client with the selected sampling frequency, as chosen in the GUI. Within this class, the GUI must also be updated to include the newly received information.

The final class to be designed is the ServerGui. This class will be used to display the graphed temperature and time information received from the Client, and allow the user some control over the displayed information, and sampling frequency. The ServerGUI will be initialised in the Server object constructor, allowing it to run as soon as the Server program is started. The GUI must then be updated from the ThreadedConnectionHandler objects in order to contain the received information. The ServerGui will contain a number of JPanel, and Graphics2D Swing components in order to correctly implement the graphing functionality required.

2.2 Implementation

The following section describes the java code implementation used for the completion of the assignment.

2.2.1 TempService

The TempService class is based on the provided DateTimeService class. This class contains all of the required information for the trasmission of the systems temperature, IP address, and current system time data. The following variables are used to store this data.

```
private String temp, samplingFreq, ipAddr;
private Calendar calendar;
private int sampleNo;
```

The temperature value is read from the system file, found on the Raspberry Pi at "/sys/class/thermal/thermal_zone'x'/temp", where the 'x' value is passed in throught the object constructor to choose the thermal zone directory, allowing for different temperature readings to be sent to the server from different clients. A StringBuffer is used to read in this value, with the output assigned to the String "temp" variable.

The time and date are set using the "Calendar.getInstance()" method, which returns an object of type Calendar. The IP Address of the system is obtained using the "InetAddress.getLocalHost().getHostAddress()" method, which returns the IP Address as a String.

There are a number of methods defined which act as getters for the parameters of the TempService objects.

The "getTemp" method returns the String "temp" variable value.

The "getTime" method returns the time value in milliseconds as a string by taking the String value of the Calendar objects time in milliseconds from the current system time in milliseconds. This allows for better ease of use when drawing lines on the ServerGui graph.

The "getSampleNum" method returns the integer value of the sampleNo variable.

The "getIP" method returns the String value of the ipAddr variable.

In order to ensure that the objects of the TempService class may be transmit from the server to the client, the class must implement "Serializable".

2.2.2 Client

The Client class is a modified version of the provided "Client.java" code. The main method of the Client class takes two command line arguements in. The first of these is the IP address of the server. The seceond acts as the index for the thermal zone, which is read from in the TempService class constructor. The constructor of the Client object calls the "connectToServer" method, passing the serverIP variable value as an arguement. This method creates a new client socket to the server at the passed in IP address, with port number 5050. The input and output stream objects are also created and associated with the client socket.

A number of methods are used in order to transmit and receive objects from the server.

The "send" method utilises the ObjectOutputStream of the Client object, using the "writeObject" and "flush" methods to transmit the input object.

The "receive" method utilises the ObjectInputStream of the Client object, using the "readObject" method to read in the object data sent from the server, and returns this object.

The "sendError" method is called if an error occurrs in transmission of an object.

Within the main method of the class, the initial "sampleNo" value is initialised as 0. An "if" statement checks if the correct number of command line arguements have been entered. If not, a warning message is output to the user on the command line, instructing them of the correct usage of the program. If the correct number of command line arguements are input, the Client object constructor is called, and a "while true" loop is entered. Within the loop, a TempService object is created, using the current sampling frequency value, current "sampleNo" value, and the second command line arguement as the inputs for the constructor. The sampleNo value is then incremented, and the TempService object is sent to the server. The selected sampling frequency is read in from the server, and the receivedFreq variable is assigned this value. If the received exitCommand value is "N" the thread then waits for the number of milliseconds given by the receivedFreq value times 1000, otherwise the Client code exits.

2.2.3 ThreadedConnectionHandler

Results

3.1 Testing Setup

Initial testing was completed using the development PC, a Ubuntu Linux system, using the "TMUX" terminal multiplexer, and Eclipse. The ThrededServer was run using either the Eclipse IDE or a TMUX terminal, while Clients were run in multiple terminals within TMUX, as shown in Figure 1. This allowed for testing using the development PC's CPU temperature, and multiple clients running within one easily readable window.

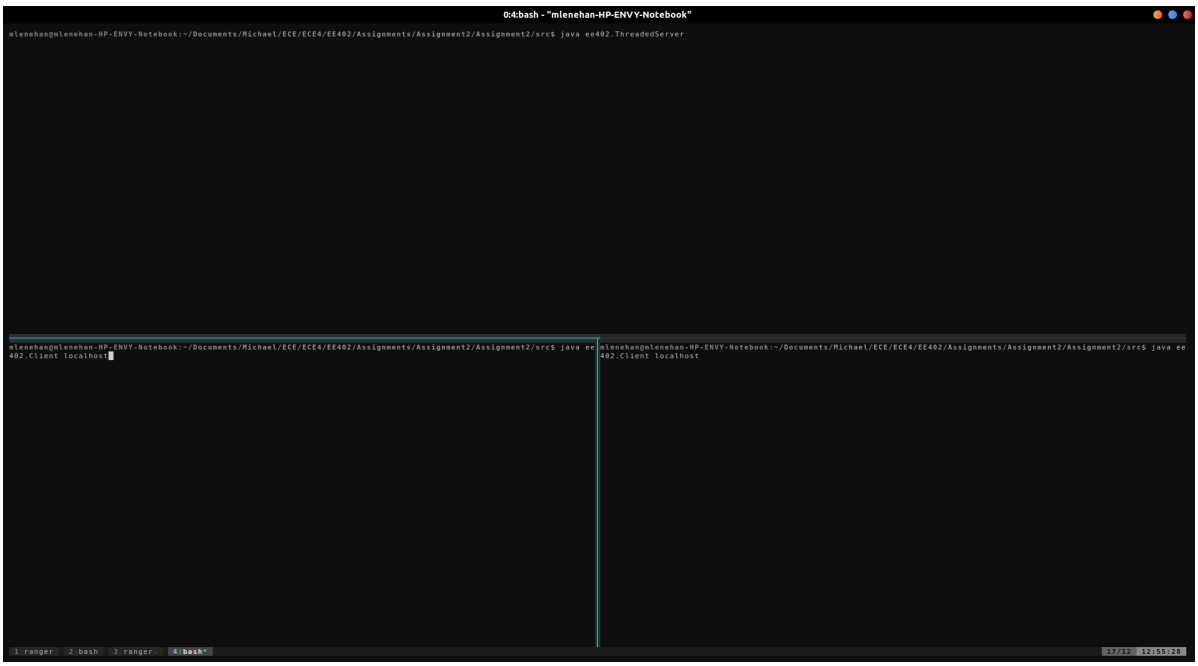

Figure 1: Testing Setup in TMUX using Two Clients

3.2 Initial CLI Testing

Initial testing within this setup yielded results which indicated that the application was working as intended, with the console output showing the stored temperature and device ID data. The output from this testing may be seen in Figure 2.

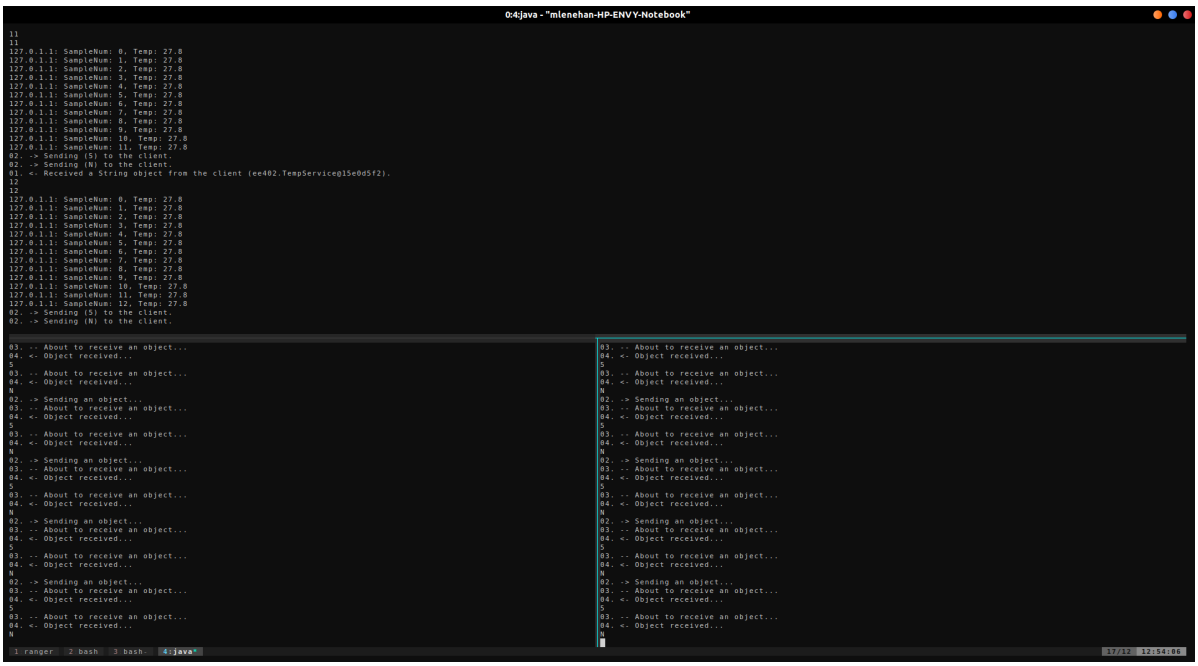

Figure 2: Initial Successful Command Line Test

The Client classes main function, and the TempService constructor were modified to take two command line arguements, the second of which is used to set the thermal zone being read from, which allows for different readings to be achieved from the same test machine. The results of this modification may be seen in Figure 3.

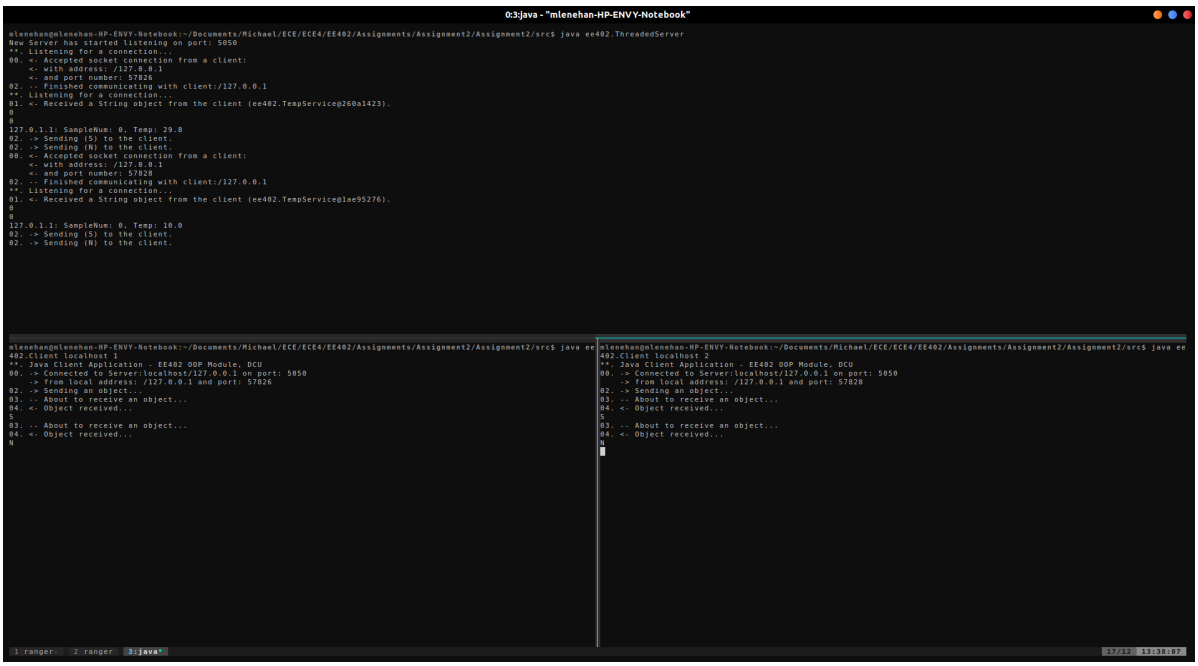

Figure 3: Successful Command Line Test with Two Input Arguements

Conclusion