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

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. At this point the Raspberry Pi LED is toggled on and off again using the provided “BasicLEDExample” code 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

The ThreadedConnectionHandler class is a modified version of the provided “Threaded-ConnectionHandler.java” code. This code is used to perform all necessary transmissions with the Client object. This allows the Server object to listen for more connecting Clients. The constructor of this class is called from the Server class, and is passed the socket information from the Client object. it then creates an ArrayList of type TempService, in which to hold all of the received TempService objects from the connected Client. The “run” method creates the ObjectInput/OutputStream objects, and places the Threaded-ConnectionHandler in a loop, waiting to receive input from the Client.

The “readInput” method reads the input object from the Client using the “readObject” method. The type of this value is then checked, and if it is a TempService object, a new TempService object is created to store the received data. An “if” statement checks the size of the Handlers ArrayList, and if it is greater than 20 (the maximum number of values to be stored), the first value of the ArrayList (at index 0) is removed, and the new TempService object is added to the end of the ArrayList. If the size of the ArrayList is less than 20, the TempService object is added to the end of the ArrayList. At this point a number of other methods are called.

The “updateGui” method takes the ArrayList as an input, passing it to the “update” method of the ServerGui class. The integer value of 2 is also passed, and works as a select in the “update” methods switch statement.

The “writeToFile” method is used to output the stored TempService objects in a file, within the users home directory (as the code was developed on Ubuntu this is required to allow for use on the Windows platform). This method utilises the “for each” loop to iterate through the ArrayList, appending the contents of the object to the string to be written to the file.

The “send” method works in the same way as the Client classes send method, utilising the ObjectOutputStream of the ThreadedConnectionHandler to write objects to the Client.

A “sendError” method is used in order to notify the Client of an error in transmission.

The “setFreq” method is used to set the sampling frequency to be sent to the Client object.

Finally, the “closeSocket” methhod is used in order to end the socket connection between the ThreadedConnectionHandler and the Client objects.

2.2.4 ThreadedServer

The ThreadedServer class is a modified version of the provided “ThreadedServer.java” code. Initially within the main method, the ThreadedServer constructor is called. This consrtuctor creates a new SererGui object. The boolean “listening” variable value is set to true, and the ServerSocket object is set to null. The server socket is then set to the portNumber value of 5050, and, if unsuccessful, will throw an error, exiting the program. If successful, a while loop is entered, which loops while the “listening” variable value is set true. The “clientSocket” value is set when the server accespts a connection from the client. The threaded handler object is then created, with the “clientSocket” passed as the arguement for the constructor. The “run” method of the ThreadedConnectionHandler is then executed using the “start” method, allowing the Server object to continue listening for Client object connections.

2.2.5 ServerGui

The ServerGui code controls the creation and setup of the applications user interface. A number of components are utilised to create this interface. The constructor of this class calls the “update” method, with a status value of “1” passed in, and an empty ArrayList of type TempService. Within the “run” method of this class, a switch statement is used to choose whether the gui is being initialised or updated. Case 1 of the switch statement creates a number of the required components, including the initial GraphPanel object, which is defined within the ServerGui class. The “mainPanel” JPanel object, “graphPanel” GraphPanel object, and “textField” JTextField objects are added to the frame. Case 2 updates the GraphPanel object with the new ArrayList of TempService values, adding it to the mainPanel, and repainting the frame.

The GraphPanel class is defined internally in the ServerGui class, extending JComponent. It draws the graph axis if there is an empty or 1 element ArrayList available. If there are more values available, it creates a Line2D object, which is drawn between the points given by the temperature and time values of the TempService objects in the ArrayList.

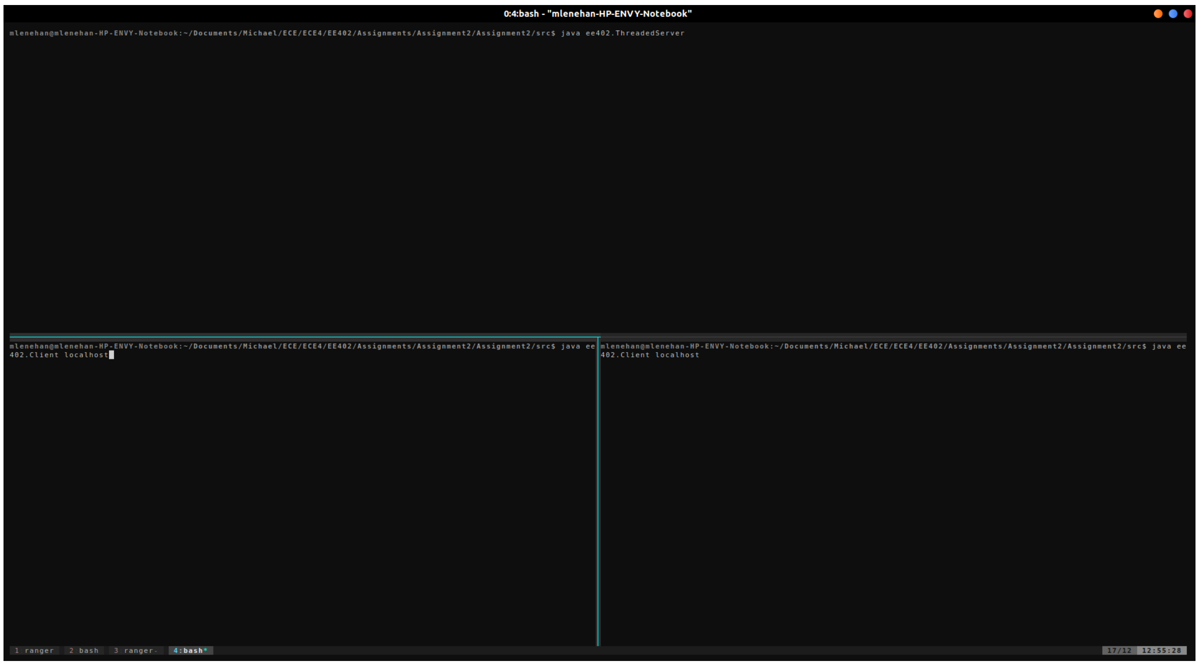
The “update” method is used in order to update the graph displayed by the gui. The “setStatus” method is called, with the input status and ArrayList being passed as arguements.

The “SwingUtilities.involeLater” method is then called, which will call the “run” method of the ServerGui when the thread is free to do so. The “setStatus” method is a synchronised method, available from all threads, which updates the “status” and “temps” values of the ServerGui object.

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.



```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee402.ThreadedServer
```

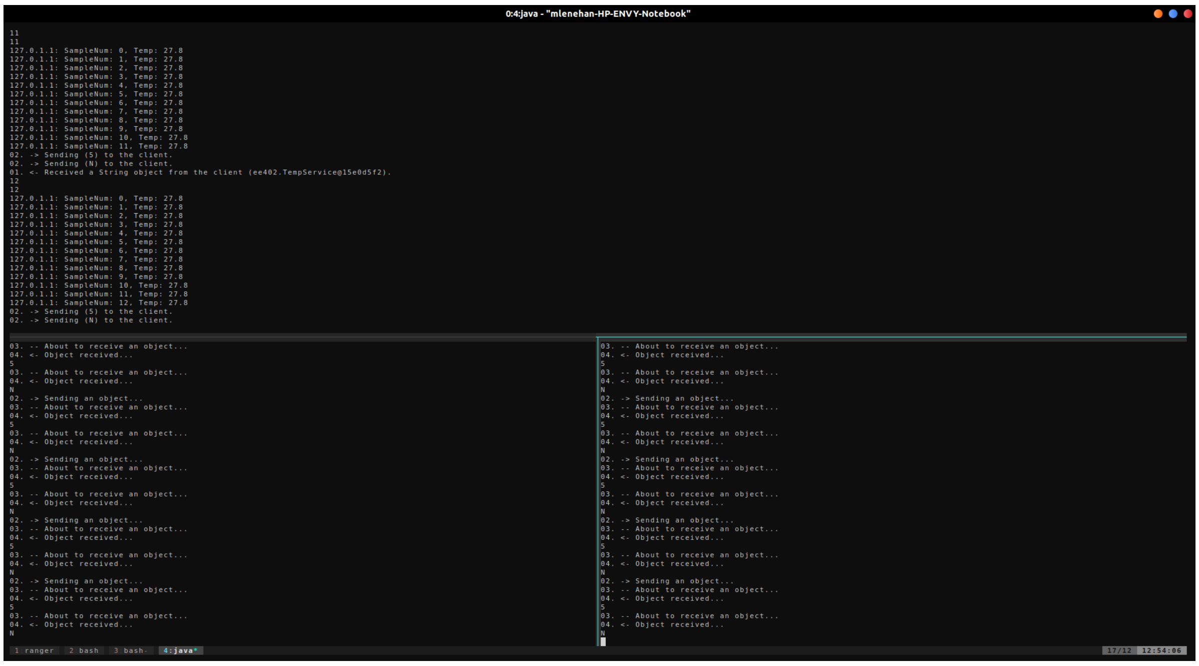
```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee
402.Client localhost
```

```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee
402.Client localhost
```

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.



```
0:4:java - "mlenehan-HP-ENVY-Notebook"
11
11
127.0.1.1: SampleNum: 0, Temp: 27.8
127.0.1.1: SampleNum: 1, Temp: 27.8
127.0.1.1: SampleNum: 2, Temp: 27.8
127.0.1.1: SampleNum: 3, Temp: 27.8
127.0.1.1: SampleNum: 4, Temp: 27.8
127.0.1.1: SampleNum: 5, Temp: 27.8
127.0.1.1: SampleNum: 6, Temp: 27.8
127.0.1.1: SampleNum: 7, Temp: 27.8
127.0.1.1: SampleNum: 8, Temp: 27.8
127.0.1.1: SampleNum: 9, Temp: 27.8
127.0.1.1: SampleNum: 10, Temp: 27.8
127.0.1.1: SampleNum: 11, Temp: 27.8
02. -> Sending (5) to the client.
02. -> Sending (N) to the client.
01. <- Received a String object from the client (ee402.TempService@15e0d5f2).
12
12
127.0.1.1: SampleNum: 0, Temp: 27.8
127.0.1.1: SampleNum: 1, Temp: 27.8
127.0.1.1: SampleNum: 2, Temp: 27.8
127.0.1.1: SampleNum: 3, Temp: 27.8
127.0.1.1: SampleNum: 4, Temp: 27.8
127.0.1.1: SampleNum: 5, Temp: 27.8
127.0.1.1: SampleNum: 6, Temp: 27.8
127.0.1.1: SampleNum: 7, Temp: 27.8
127.0.1.1: SampleNum: 8, Temp: 27.8
127.0.1.1: SampleNum: 9, Temp: 27.8
127.0.1.1: SampleNum: 10, Temp: 27.8
127.0.1.1: SampleNum: 11, Temp: 27.8
127.0.1.1: SampleNum: 12, Temp: 27.8
02. -> Sending (5) to the client.
02. -> Sending (N) to the client.
```

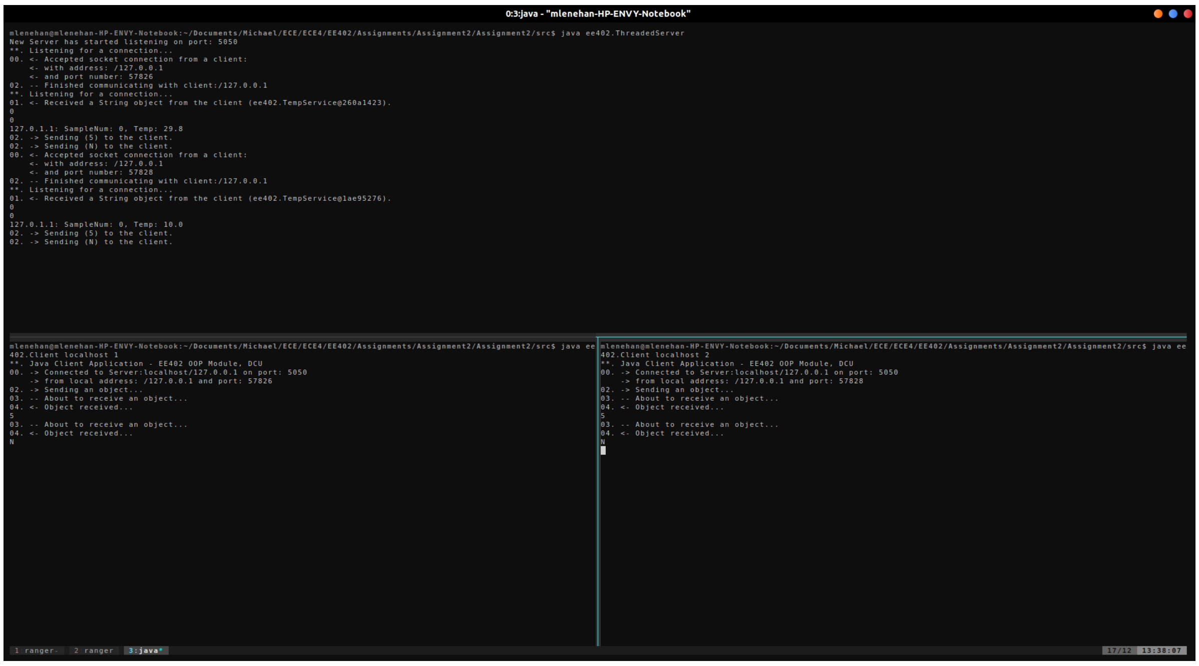
```
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
```

```
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
```

```
1 ranger  2 bash  3 bash-  4:java*                                17/12  12:54:06
```

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.



```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee402.ThreadedServer
New Server has started listening on port: 5050
**. Listening for a connection...
00. <- Accepted socket connection from a client:
    <- with address: /127.0.0.1
    <- and port number: 57826
02. -- Finished communicating with client:/127.0.0.1
**. Listening for a connection...
01. <- Received a String object from the client (ee402.TempService@260a1423).
0
0
127.0.1.1: SampleNum: 0, Temp: 29.8
02. -> Sending (5) to the client.
02. -> Sending (N) to the client.
00. <- Accepted socket connection from a client:
    <- with address: /127.0.0.1
    <- and port number: 57828
02. -- Finished communicating with client:/127.0.0.1
**. Listening for a connection...
01. <- Received a String object from the client (ee402.TempService@1ae95276).
0
0
127.0.1.1: SampleNum: 0, Temp: 10.0
02. -> Sending (5) to the client.
02. -> Sending (N) to the client.
```

```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee
402.Client localhost 1
**. Java Client Application - EE402 OOP Module, DCU
00. -> Connected to Server:localhost/127.0.0.1 on port: 5050
    -> from local address: /127.0.0.1 and port: 57826
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
```

```
mlenehan@mlenehan-HP-ENVY-Notebook:~/Documents/Michael/ECE/ECE4/EE402/Assignments/Assignment2/Assignment2/src$ java ee
402.Client localhost 2
**. Java Client Application - EE402 OOP Module, DCU
00. -> Connected to Server:localhost/127.0.0.1 on port: 5050
    -> from local address: /127.0.0.1 and port: 57828
02. -> Sending an object...
03. -- About to receive an object...
04. <- Object received...
5
03. -- About to receive an object...
04. <- Object received...
N
```

Figure 3: Successful Command Line Test with Two Input Arguements

3.3 Graphical User Interface

Unfortunately the Server Gui class is not functional as intended. A NullPointerException is thrown when the ArrayList is passed to the GraphPanels "paintComponent" method, however, at certain times, it updates with incorrectly read values, drawing a line to the graph, and then proceeds to throw NullPointerExceptions. This may be due to a misunderstaning in the functionality of the "paintComponent" method, or the incorrect initialisation of the ServerGui from within the ThreadedServer class, instead of using the "start" method. As such, the gui is essentially non-functional and has been commented out for the testing of this code.

The inconsistent output from running the ServerGui code may be seen in the attached log file.

3.4 File Writer

On returning to the command line application, the additional functionality is a file writer, which places the required output data in a text file. This gives an opportunity to view the output from the application in a readable form, without a functioning gui. It also allows for a saved output to be viewed at a later stage if necessary, e.g. for system monitoring purposes.

Conclusion

Unfortunately due to issues in understanding the GUI elements of this assignment, many of the points could not be completed correctly, the graph is not displayed to the user, and as such the application works on the command line only.

As a command line application, the values are transmit and stored correctly. The additional functionality of saving the temperature values works as intended, and the light on the Raspberry Pi flashes as intended.