**DCU School of Electronic Engineering**
**Assignment Submission**

| | |
|---|---|
| **Student Name(s):** | **Radwan Duadu** |
| **Student Number(s):** | **14342606** |
| **Programme:** | **B.Eng. in Electronic Engineering** |
| **Project Title:** | **Assignment 2** |
| **Module code:** | **EE402** |
| **EMAIL** | **Radwan.duadu2@mail.dcu.ie** |
| **Lecturer:** | **Derek** |
| **Project Due Date:** | |

**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 and acknowledged within the text of my work. I understand that plagiarism, 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.pdfand/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:**_*Radwan duadu*_____

# *ASSIGNMENT 2*

## Design

The assignment was split into three different parts server, client and GUI (graphical user interface), the code was designed so that the client can connect to multiple servers and using the GUI data can be acquired in real time from the servers. The servers are set up on the raspberry PI.

In the code it was designed so that the server contains a server connection handler which allows for multiple server connections to be established using multiple threads, this was not the case for the client the client contains only one thread but when the constructer is run it predefines how many sockets are opened on the client which allows the client to connect to multiple servers. The data is then used to represent in a more user friendly on the GUI.

## Implementation

The code starts by running the APP.java which sets the GUI parameters and running the Display.java the display java method will add components to the GUI and set up connection between client and server.

The code to connect to different servers can be seen below.

```
the method expects the IP address of the server - the port is fixed
    private boolean connectToServer(String serverIP,int portNumber, int offset) {
       try {
        // open a new socket to the server
       this.socket = new Socket(serverIP,portNumber);
       //this.os = new ObjectOutputStream(this.socket.getOutputStream());
       //this.is = new ObjectInputStream(this.socket.getInputStream());
       System.out.println("00. -> Connected to Server:" + this.socket.getInetAddress()
                                + " on port: " + this.socket.getPort());
       System.out.println("    -> from local address: " + this.socket.getLocalAddress()
                                + " and port: " + this.socket.getLocalPort());

        if(socket != null && gui != null) {
                    con = new ThreadededClientConnectionHandler(socket, gui, offset);
                    con.start();

       }
        catch (Exception e) {
         System.out.println("XX. Failed to Connect to the Server at port: " +portNumber);
         System.out.println("    Exception: " + e.toString());
         return false;
         }
              return true;
    }
```

As can be seen above multiple sockets are initialised depending on the number of input servers inputted to the client constructor.

The GUI then contains a textbox which will take in a number argument and this dictates the speed in which the request to retrieve the temp is taken from the different servers.

A button is then used to take the argument retrieved from the textbox and then the function getSampleRate() this function will send the request to the client to retrieve the temperature from the different servers and the result is then added to different lists.

This is done so that each port will contain a list of temp frequency only the 20 most recent values are left any older values will be removed ensuring an accurate reading of the temp.

The code for the tempfreq() function can be seen below.

```
public void tempList() {
		int tempValue = (int) getTempValue() + offset;
		list.add(tempValue);
		if(list.size() > 20) {
			try {l20 = list.subList(list.size()-20, list.size());}
			catch (IndexOutOfBoundsException e)
			{	e.printStackTrace();
				l20 = list;
			} catch (NullPointerException e) {
				l20 = new ArrayList<Integer>();
			}
		}
		else {l20 = list;}
		System.out.println(l20);

	}

	//sets the sample rate
	public int setSampleRate() {
		int sampleRate = gui.getSampleRate();
		while (sampleRate != 0) {
			this.tempList();
			gui.Arraytemp(l20);
			try {
				Thread.sleep(sampleRate);
			}
			catch (InterruptedException e) {
				e.printStackTrace();
			}
			//updates the sample rate value
			if(sampleRate!= gui.getSampleRate()) {
				sampleRate = gui.getSampleRate();

			}
		}
		System.out.println(sampleRate);
		return sampleRate;
	}

	//specifically gets just temperature value, gets the TempService class
	//from ThreadedConnectionHandler in the server
	public double getTempValue() {
		String theTempValueCommand = "GetTemp";
		String theTempValue = "";
		this.send(theTempValueCommand);
```

```
            try{
                  theTempValue = (String) receive();
            }
            catch (Exception e){
                  System.out.println(e);
            }
            double tempvalue = Double.parseDouble(theTempValue)/1000;
            return tempvalue;

      }
```

The code to run the setSampleRate() is found in the Sample Rate button this button will run return most recent 20 values.

Code for the button can be found below.

```
displaytemp.addActionListener(new ActionListener() {
//panel 3 - sample rate text field, sample rate button, clear text field button
            JPanel p3 = new JPanel();
            p3.setLayout(new GridLayout(1,3));
            TextSampleRate = new JTextField(5);
            p3.add(this.TextSampleRate);
            SampleRate = new JButton("Sample Rate");
            p3.add(this.SampleRate);
            ClearSampleRate = new JButton("Clear text");
            p3.add(this.ClearSampleRate);
            SampleRate.addActionListener(this);
            ClearSampleRate.addActionListener(this);
```

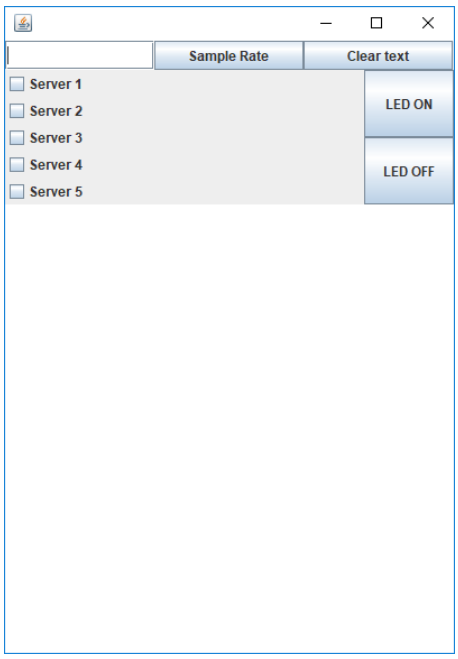
The check boxes will only draw when selected and it will not draw when unselected

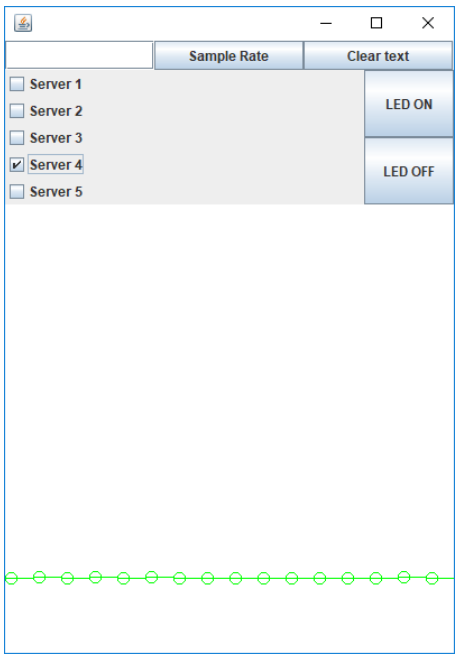
The code for drawing the graph can be seen below.

```
/paints the graph
            public void paint(Graphics g) {

            for(int j=0; j <5;j++) {
                  if(boxes[j].isSelected() == true) {
                  Graphics2D g2d = (Graphics2D) g;

                  g2d.setColor(color[j]);
                  if(last20 != null) {
                        for(int i=0; i<last20.size(); i++){
                              int y1 = last20.get(i);
                              g2d.drawLine(i*25, 200+y1, (i+1)*25, 200+y1);
                              g2d.drawOval(i*25,195+ y1, 2*radius, 2*radius);

                        }
                  }
                  }
            }
            }
```

As can be seen the code draws the graph from each temp point to the next making it possible to see when the temperature changes in the graph.

The final method was a method I made up to control the LED state and is displayed in the GUI application below.




Code can be seen below.

**APP.java**

```
package ee402;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

public class APP extends Canvas{

	private static final int WIDTH = 400;
	private static final int HEIGHT = 400;
	public List<Integer> last20 = new ArrayList<Integer>();
	private JCheckBox boxes[] = new JCheckBox[5];
	private Color[] color = new Color[5];
	static int radius = 5;

	public APP() {
		this.setSize(WIDTH,HEIGHT);
		this.setBackground(Color.WHITE);
		 color[0] = Color.red;
	     color[1] = Color.blue;
	     color[2] = Color.yellow;
	     color[3] = Color.green;
	     color[4] = Color.ORANGE;
		 this.repaint();
	}
```

```
	public void UpdateTemperature (List<Integer> last20,JCheckBox boxes[]) {
		this.last20 = last20;
		for(int i=0;i<5;i++) {
			this.boxes[i] = boxes[i];
		}
		this.repaint();
	}

	//paints the graph
		public void paint(Graphics g) {

		for(int j=0; j <5;j++) {
			if(boxes[j].isSelected()) {
			Graphics2D g2d = (Graphics2D) g;

			g2d.setColor(color[j]);
			if(last20 != null) {
				for(int i=0; i<last20.size(); i++){
					int y1 = last20.get(i);
					g2d.drawLine(i*25, 200+y1, (i+1)*25, 200+y1);
					g2d.drawOval(i*25,195+ y1, 2*radius, 2*radius);

				}
			}
			}
		}
		}

}
```

**TempAndDateGui.java**

```
package ee402;

import java.awt.*;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

public class TempAndDateGui extends JFrame implements ActionListener{

	private JCheckBox boxes[] = new JCheckBox[5];
	private List<Integer> last20 = new ArrayList<Integer>();
	private JButton SampleRate, ClearSampleRate;
	private JButton LEDON, LEDOFF;
	private int SampleValue= 1000;
	private String ledcommand= "";
	private JTextField TextSampleRate;
	private APP canvas;
	private LEDcontroller led = new LEDcontroller();

	public TempAndDateGui() {
		super();
		Container cont = getContentPane();
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

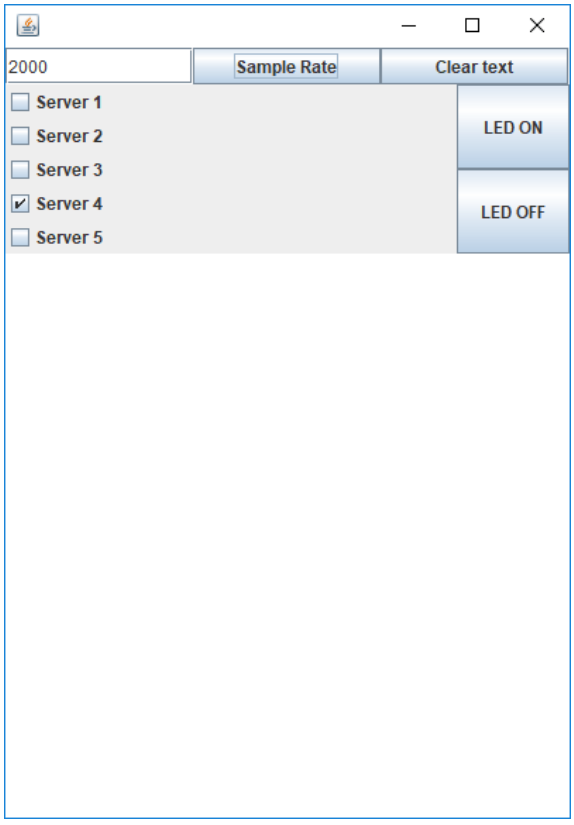
```
            //panel 3 - sample rate text field, sample rate button, clear text
field button
            JPanel p3 = new JPanel();
            p3.setLayout(new GridLayout(1,3));
            TextSampleRate = new JTextField(5);
            p3.add(this.TextSampleRate);
            SampleRate = new JButton("Sample Rate");
            p3.add(this.SampleRate);
            ClearSampleRate = new JButton("Clear text");
            p3.add(this.ClearSampleRate);
            SampleRate.addActionListener(this);
            ClearSampleRate.addActionListener(this);

            JPanel p4 = new JPanel();
            p4.setLayout(new GridLayout(2,1));
            LEDON =new JButton("LED ON");
            p4.add(this.LEDON);
            LEDOFF = new JButton("LED OFF");
            p4.add(this.LEDOFF);

            JPanel p1 = new JPanel();
            p1.setLayout(new BorderLayout());
            canvas = new APP();
            p1.add(this.canvas, BorderLayout.SOUTH);

            //panel 2 - server check boxes
            JPanel p2 = new JPanel();
            p2.setLayout(new GridLayout(5,1));
            for(int i = 0; i<5; i++) {
                  boxes[i] = new JCheckBox("Server "+ (i+1));
                  boxes[i].addActionListener(this);
                  p2.add(this.boxes[i]);
            }

            //layout of all panels
            cont.setLayout(new BorderLayout());
            cont.add(p3, BorderLayout.NORTH);
            cont.add(p2, BorderLayout.WEST);
            cont.add(p4, BorderLayout.EAST);
            cont.add(p1, BorderLayout.SOUTH);

            this.pack();
            this.setVisible(true);


      }
      public int getSampleRate() {
            int a = this.SampleValue;
            return a;
      }
      public void Arraytemp (List<Integer> last2) {
            this.last20 = last2;
            canvas.UpdateTemperature(last20,boxes);
      }


      @Override
      public void actionPerformed(ActionEvent e) {
```

```
            System.out.println("An action event has occured");
            //takes the string input in the text field
            int sampleNum  = (new
Integer(this.TextSampleRate.getText())).intValue();
            SampleValue = sampleNum;
            //clears the text field
            if(e.getSource().equals(ClearSampleRate)) {
                  this.TextSampleRate.setText("");
            }else if(e.getSource().equals(LEDON)) {
                  ledcommand= "LEDON";
            }else if(e.getSource().equals(LEDOFF)) {
                  ledcommand= "LEDOFF";
            }



      }


}
```

```
import java.net.*;
import java.text.SimpleDateFormat;
import java.io.*;

public class Client {

      private Socket socket = null;
      private TempAndDateGui gui;
      private ThreadededClientConnectionHandler con;
      private APP canvas;


public Client() {
      gui = new TempAndDateGui();
    }

 // the method expects the IP address of the server - the port is fixed
    private boolean connectToServer(String serverIP,int portNumber, int offset)
{
      try { // open a new socket to the server
                  this.socket = new Socket(serverIP,portNumber);
                  //this.os = new
ObjectOutputStream(this.socket.getOutputStream());
                  //this.is = new
ObjectInputStream(this.socket.getInputStream());
                  System.out.println("00. -> Connected to Server:" +
this.socket.getInetAddress()
                              + " on port: " + this.socket.getPort());
                  System.out.println("    -> from local address: " +
this.socket.getLocalAddress()
                              + " and port: " + this.socket.getLocalPort());

                  if(socket != null && gui != null) {
                              con = new
ThreadededClientConnectionHandler(socket, gui, offset);
                              con.start();
                                                }
```

```
        }
          catch (Exception e) {
                System.out.println("XX. Failed to Connect to the Server at port: "
+ portNumber);
                System.out.println("    Exception: " + e.toString());
                return false;
          }

                return true;
    }
    public static void main(String args[]) throws InterruptedException
    {
       System.out.println("**. Java Client Application - EE402 OOP Module,
DCU");
                Client theApp = new Client();
                theApp.connectToServer("192.168.0.31", 5051, 0);
                theApp.connectToServer("192.168.0.31", 5052, 40);
                theApp.connectToServer("192.168.0.31", 5053, 80);
                theApp.connectToServer("192.168.0.31", 5054, 120);
                theApp.connectToServer("192.168.0.31", 5055, 160);

    }
}
```

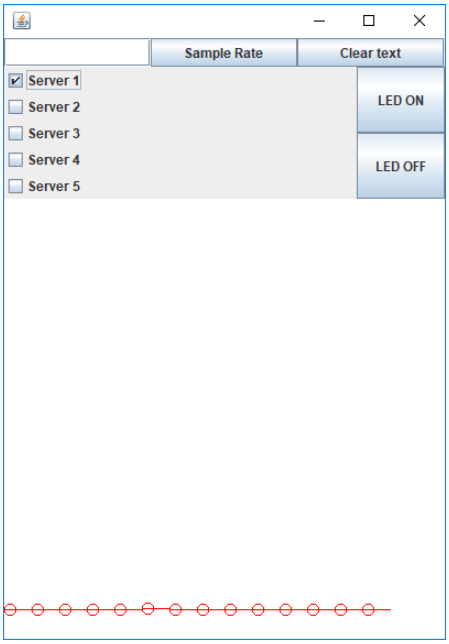
## Documentation



As can be seen above the code is now running all the commands are carried out 5 times this is representing the 5 clients sending and receiving information to the 5 servers even though some server and client response are faster all 5 ports are being used.

```
02. -> Sending an object...
02. -> Sending an object...
03. -- About to receive an object...
02. -> Sending an object...
02. -> Sending an object...
03. -- About to receive an object...
03. -- About to receive an object...
02. -> Sending an object...
03. -- About to receive an object...
03. -- About to receive an object...
04. <- Object received...
[53, 53, 53, 53, 53, 52, 53, 53, 53, 53, 53, 52, 52, 52, 53, 52, 53, 53, 52, 53]
04. <- Object received...
[132, 133, 133, 133, 133, 132, 133, 133, 133, 132, 133, 133, 133, 133, 133, 133,
133, 132, 133, 133]
04. <- Object received...
[213, 213, 213, 213, 212, 213, 213, 213, 212, 213, 212, 213, 212, 213, 212, 213,
213, 212, 213, 213]
04. <- Object received...
[92, 92, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 93, 92, 93, 93]
04. <- Object received...
[172, 173, 173, 173, 173, 173, 173, 173, 173, 173, 173, 173, 173, 173, 172, 173,
173, 172, 173, 173]
02. -> Sending an object...
03. -- About to receive an object...
02. -> Sending an object...
02. -> Sending an object...
03. -- About to receive an object...
03. -- About to receive an object...
02. -> Sending an object...
02. -> Sending an object...
03. -- About to receive an object...
03. -- About to receive an object...
04. <- Object received...
[53, 53, 53, 53, 52, 53, 53, 53, 53, 53, 52, 52, 52, 53, 52, 53, 53, 52, 53, 52]
04. <- Object received...
[133, 133, 133, 133, 132, 133, 133, 133, 132, 133, 133, 133, 133, 133, 133, 133,
132, 133, 133, 133]
04. <- Object received...
[213, 213, 213, 212, 213, 213, 213, 212, 213, 212, 213, 212, 213, 212, 213, 213,
212, 213, 213, 213]
04. <- Object received...
```

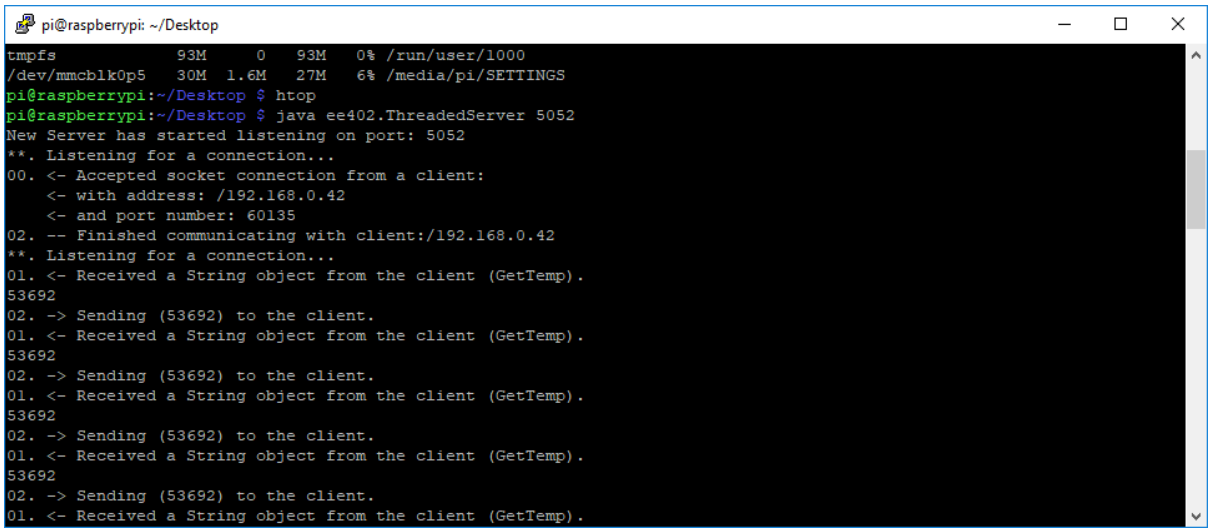
Above is an example of what occurs the code is run in a loop and the list increases in size by one until it hits 20 and discards the oldest values.




As can be seen when each checkbox was unchecked and then the pressing of the graph button only the checkboxes that contained the checkboxes that were checked would be drawn. It also updates as time passes according to the sampling rate.

The final buttons were used to control the LED light on the server

The final part of the implantation is the set up of the servers on the raspberry pi which can be seen below.




## SOURCE-CODE

**ThreadedConnectionHandler.java**

```
public class ThreadedConnectionHandler extends Thread
{
    private Socket clientSocket = null;                    // Client socket
object
    private ObjectInputStream is = null;                   // Input stream
```

```
    private ObjectOutputStream os = null;                    // Output stream
    private DateTimeService theDateService;
    private TempService theTempService;
    private LEDcontroller led;

      // The constructor for the connection handler
    public ThreadedConnectionHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
        //Set up a service object to get the current date and time
        theDateService = new DateTimeService();
        theTempService = new TempService();
        led = new LEDcontroller();
    }

    // Will eventually be the thread execution method - can't pass the exception
back
    public void run() {
         try {
            this.is = new ObjectInputStream(clientSocket.getInputStream());
            this.os = new ObjectOutputStream(clientSocket.getOutputStream());
            while (this.readCommand()) {}
         }
         catch (IOException e)
         {
             System.out.println("XX. There was a problem with the Input/Output
Communication:");
            e.printStackTrace();
         }
    }

    // Receive and process incoming string commands from client socket
    private boolean readCommand() {
        String s = null;
        try {
            s = (String) is.readObject();
        }
        catch (Exception e){    // catch a general exception
             this.closeSocket();
            return false;
        }
        System.out.println("01. <- Received a String object from the client (" +
s + ").");

        // At this point there is a valid String object
        // invoke the appropriate function based on the command
        if (s.equalsIgnoreCase("GetDate")){
            this.getDate();
        }
        else if (s.equalsIgnoreCase("GetTemp")) {
             this.getTemp();
        }
        else if (s.equalsIgnoreCase("LEDON")) {
             this.LEDon();
        }
        else if (s.equalsIgnoreCase("LEDOFF")) {
             this.LEDoff();
        }

        else {
```

```
            this.sendError("Invalid command: " + s);
        }
        return true;
    }

    // Use our custom DateTimeService Class to get the date and time
    private void getDate() {     // use the date service to get the date
        String currentDateTimeText = theDateService.getDateAndTime();
        this.send(currentDateTimeText);
    }
 // turn on led
    private void LEDon() {// use the date service to get the date
        String LEDstat = "LED turned ON";
        led.LEDON();
        this.send(LEDstat);
    }
 // turn on led
    private void LEDoff() {      // use the date service to get the date
        String LEDstat = "LED turned OFF";
        led.LEDOFF();
        this.send(LEDstat);
    }
    private void getTemp(){
        String currentTempText = theTempService.getTemperature();
        this.send(currentTempText);
    }
    // Send a generic object back to the client
    private void send(Object o) {
        try {
            System.out.println("02. -> Sending (" + o +") to the client.");
            this.os.writeObject(o);
            this.os.flush();
        }
        catch (Exception e) {
            System.out.println("XX." + e.getStackTrace());
        }
    }

    // Send a pre-formatted error message to the client
    public void sendError(String message) {
        this.send("Error:" + message); //remember a String IS-A Object!
    }

    // Close the client socket
    public void closeSocket() { //gracefully close the socket connection
        try {
            this.os.close();
            this.is.close();
            this.clientSocket.close();
        }
        catch (Exception e) {
            System.out.println("XX. " + e.getStackTrace());
        }
    }
}
```

**ThreadedServer.java**

```
public class ThreadedServer
{
	public ThreadedServer(String port){}



	public void com(String port) {
		 int portNumber = Integer.parseInt(port);
		 boolean listening = true;
	  ServerSocket serverSocket = null;

	  // Set up the Server Socket
	  try
	  {
	      serverSocket = new ServerSocket(portNumber);
	      System.out.println("New Server has started listening on port: " +
portNumber );
	  }
	  catch (IOException e)
	  {
	      System.out.println("Cannot listen on port: " + portNumber + ",
Exception: " + e);
	      System.exit(1);
	  }

	  // Server is now listening for connections or would not get to this
point
	  while (listening) // almost infinite loop - loop once for each client
request
	  {
	      Socket clientSocket = null;
	      try{
	       System.out.println("**. Listening for a connection...");
	          clientSocket = serverSocket.accept();
	          System.out.println("00. <- Accepted socket connection from a
client: ");
	          System.out.println("    <- with address: " +
clientSocket.getInetAddress().toString());
	          System.out.println("    <- and port number: " +
clientSocket.getPort());
	      }
	      catch (IOException e){
	          System.out.println("XX. Accept failed: " + portNumber + e);
	          listening = false;   // end the loop - stop listening for
further client requests
	      }

	      ThreadedConnectionHandler con = new
ThreadedConnectionHandler(clientSocket);
	      con.start();
	      System.out.println("02. -- Finished communicating with client:" +
clientSocket.getInetAddress().toString());
	  }
	  // Server is no longer listening for client connections - time to shut
down.
	  try
	  {
	      System.out.println("04. -- Closing down the server socket
gracefully.");
```

```
            serverSocket.close();
        }
        catch (IOException e)
        {
            System.err.println("XX. Could not close server socket. " +
e.getMessage());
        }
    }
      public static void main(String args[]) {
            if(args.length==1){
                  ThreadedServer theApp = new ThreadedServer(args[0]);
                  theApp.com(args[0]);
            }
            else
            {
                  System.out.println("Error: you must provide the address of
the server");
                  System.out.println("Usage is:  java Client x.x.x.x  (e.g.
java Client 192.168.7.2)");
                  System.out.println("      or:  java Client hostname (e.g.
java Client localhost)");
            }
            System.out.println("**. End of Application.");
      }
}
```

**DateTimeServer.java**

```
public class DateTimeService
{
   private Calendar calendar;

   //constructor creates the Calendar object, could use the constructor:
   //   Calendar(TimeZone zone, Locale aLocale) to explicitly specify
   //    the time zone and locale
   public DateTimeService()
   {
        this.calendar = Calendar.getInstance();
   }

   //method returns date/time as a formatted String object
   public String getDateAndTime()
   {
       Date d = this.calendar.getTime();

     return " " + d.toString();
   }
                                          }
```

**LEDcontroller.java**

```
package ee402;

import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedReader;
```

```
public class LEDcontroller {
	public static String PATH ="/sys/class/leds/led0/trigger";

	public void LEDON() {
		try {
		BufferedWriter bw = new BufferedWriter(new FileWriter(PATH));
		 bw.write("none");
		 bw.close();
		 bw = new BufferedWriter(new FileWriter(PATH+"/brightness"));
		 bw.write(1);
		 bw.close();  }
		catch(FileNotFoundException e){
			System.out.println("Unable to open file");
			e.printStackTrace();
		}
		catch(IOException e){
			System.out.println("Error write file");
			e.printStackTrace();
		}
		 }

	public void LEDOFF() {
		try {
		BufferedWriter bw = new BufferedWriter(new FileWriter(PATH));
		bw.write("none");
		bw.close();
		bw = new BufferedWriter(new FileWriter(PATH+"/brightness"));
		bw.write(0);
		bw.close();
	}
	catch(FileNotFoundException e){
		System.out.println("Unable to open file");
		e.printStackTrace();
	}
	catch(IOException e){
		System.out.println("Error write file");
		e.printStackTrace();
	}
		}

}
```

**TEMPService.java**

```
package ee402;


import java.io.*;
public class TempService
{
	private String filename = "/sys/class/thermal/thermal_zone0/temp";
	private String line = null;
	private String tempValue;

	public String getTemperature(){
		try{
			BufferedReader br = new BufferedReader(new
FileReader(filename));
			while((line = br.readLine()) != null) {
			   System.out.println(line);
			   tempValue = line;
```

```
                }
                br.close();
            }
            catch(FileNotFoundException e){
                System.out.println("Unable to open file");
                e.printStackTrace();
            }
            catch(IOException e){
                System.out.println("Error reading file");
                e.printStackTrace();
            }
            return  tempValue;
        }
}
```

**APP.java**

```
package ee402;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

public class APP extends Canvas{

        private static final int WIDTH = 400;
        private static final int HEIGHT = 400;
        public List<Integer> last20 = new ArrayList<Integer>();
        private JCheckBox boxes[] = new JCheckBox[5];
        private Color[] color = new Color[5];
        static int radius = 5;

        public APP() {
            this.setSize(WIDTH,HEIGHT);
            this.setBackground(Color.WHITE);
             color[0] = Color.red;
           color[1] = Color.blue;
           color[2] = Color.yellow;
           color[3] = Color.green;
           color[4] = Color.ORANGE;
            this.repaint();
        }


        public void UpdateTemperature (List<Integer> last20,JCheckBox boxes[]) {
            this.last20 = last20;
            for(int i=0;i<5;i++) {
                this.boxes[i] = boxes[i];
            }
            this.repaint();
        }

        //paints the graph
            public void paint(Graphics g) {

            for(int j=0; j <5;j++) {
                if(boxes[j].isSelected()) {
                Graphics2D g2d = (Graphics2D) g;
```

```
                    g2d.setColor(color[j]);
                    if(last20 != null) {
                          for(int i=0; i<last20.size(); i++){
                                int y1 = last20.get(i);
                                g2d.drawLine(i*25, 200+y1, (i+1)*25, 200+y1);
                                g2d.drawOval(i*25,195+ y1, 2*radius, 2*radius);

                          }
                    }
                    }
             }
             }

}
```

**TempAndDateGui.java**

```
package ee402;

import java.awt.*;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;

public class TempAndDateGui extends JFrame implements ActionListener{

       private JCheckBox boxes[] = new JCheckBox[5];
       private List<Integer> last20 = new ArrayList<Integer>();
       private JButton SampleRate, ClearSampleRate;
       private JButton LEDON, LEDOFF;
       private int SampleValue= 1000;
       private String ledcommand= "";
       private JTextField TextSampleRate;
       private APP canvas;
       private LEDcontroller led = new LEDcontroller();

       public TempAndDateGui() {
              super();
              Container cont = getContentPane();
              setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

              //panel 3 - sample rate text field, sample rate button, clear text
field button
              JPanel p3 = new JPanel();
              p3.setLayout(new GridLayout(1,3));
              TextSampleRate = new JTextField(5);
              p3.add(this.TextSampleRate);
              SampleRate = new JButton("Sample Rate");
              p3.add(this.SampleRate);
              ClearSampleRate = new JButton("Clear text");
              p3.add(this.ClearSampleRate);
              SampleRate.addActionListener(this);
              ClearSampleRate.addActionListener(this);

              JPanel p4 = new JPanel();
              p4.setLayout(new GridLayout(2,1));
```

```
            LEDON =new JButton("LED ON");
            p4.add(this.LEDON);
            LEDOFF = new JButton("LED OFF");
            p4.add(this.LEDOFF);

            JPanel p1 = new JPanel();
            p1.setLayout(new BorderLayout());
            canvas = new APP();
            p1.add(this.canvas, BorderLayout.SOUTH);

            //panel 2 - server check boxes
            JPanel p2 = new JPanel();
            p2.setLayout(new GridLayout(5,1));
            for(int i = 0; i<5; i++) {
                  boxes[i] = new JCheckBox("Server "+ (i+1));
                  boxes[i].addActionListener(this);
                  p2.add(this.boxes[i]);
            }

            //layout of all panels
            cont.setLayout(new BorderLayout());
            cont.add(p3, BorderLayout.NORTH);
            cont.add(p2, BorderLayout.WEST);
            cont.add(p4, BorderLayout.EAST);
            cont.add(p1, BorderLayout.SOUTH);

            this.pack();
            this.setVisible(true);


      }
      public int getSampleRate() {
            int a = this.SampleValue;
            return a;
      }
      public void Arraytemp (List<Integer> last2) {
            this.last20 = last2;
            canvas.UpdateTemperature(last20,boxes);
      }


      @Override
      public void actionPerformed(ActionEvent e) {
            System.out.println("An action event has occured");
            //takes the string input in the text field
            int sampleNum  = (new
Integer(this.TextSampleRate.getText())).intValue();
            SampleValue = sampleNum;
            //clears the text field
            if(e.getSource().equals(ClearSampleRate)) {
                  this.TextSampleRate.setText("");
            }else if(e.getSource().equals(LEDON)) {
                  ledcommand= "LEDON";
            }else if(e.getSource().equals(LEDOFF)) {
                  ledcommand= "LEDOFF";
            }
```

```
	}


}
```

**Client.java**

```
import java.net.*;
import java.text.SimpleDateFormat;
import java.io.*;

public class Client {

	private Socket socket = null;
	private TempAndDateGui gui;
	private ThreadededClientConnectionHandler con;
	private APP canvas;


    public Client() {
	   gui = new TempAndDateGui();
    }

 // the method expects the IP address of the server - the port is fixed
    private boolean connectToServer(String serverIP,int portNumber, int offset)
{
	try { // open a new socket to the server
                    this.socket = new Socket(serverIP,portNumber);
                    //this.os = new
ObjectOutputStream(this.socket.getOutputStream());
                    //this.is = new
ObjectInputStream(this.socket.getInputStream());
                    System.out.println("00. -> Connected to Server:" +
this.socket.getInetAddress()
                                    + " on port: " + this.socket.getPort());
                    System.out.println("    -> from local address: " +
this.socket.getLocalAddress()
                                    + " and port: " + this.socket.getLocalPort());

                    if(socket != null && gui != null) {
                                    con = new
ThreadededClientConnectionHandler(socket, gui, offset);
                                    con.start();
                                                    }

	}
	  catch (Exception e) {
	       System.out.println("XX. Failed to Connect to the Server at port: "
+ portNumber);
	       System.out.println("    Exception: " + e.toString());
	       return false;
	  }

	       return true;
    }



    public static void main(String args[]) throws InterruptedException
    {
```

```
        System.out.println("**. Java Client Application - EE402 OOP Module,
DCU");
                Client theApp = new Client();
                theApp.connectToServer("192.168.0.31", 5051, 0);
                theApp.connectToServer("192.168.0.31", 5052, 40);
                theApp.connectToServer("192.168.0.31", 5053, 80);
                theApp.connectToServer("192.168.0.31", 5054, 120);
                theApp.connectToServer("192.168.0.31", 5055, 160);

        }
}
```

**ThreadededClientConnectionHandler.java**

```
public class ThreadededClientConnectionHandler extends Thread{
        private Socket Socket = null;                           // Client socket
object
        private ObjectInputStream is = null;                    // Input stream
        private ObjectOutputStream os = null;                   // Output stream
        private TempAndDateGui gui;
        private ArrayList<Integer> list = new ArrayList<Integer>();
        private List<Integer> l20 = null; //to store the last 20 temperature
values
        private APP canvas;
        private int offset; // temperature offset


        //constructor for Client connection handler
        public ThreadededClientConnectionHandler(Socket Socket, TempAndDateGui
gui, int offset) throws IOException {
                this.Socket = Socket;
                this.gui = gui;
                this.offset = offset;
                this.os = new ObjectOutputStream(this.Socket.getOutputStream());
                this.is= new ObjectInputStream(this.Socket.getInputStream());

        }

        public void run() {
                while(true) {
                        this.setSampleRate(); // sets the sample rate (1s)
                }
        }

        //gets the temperature values and puts the last 20
        public void tempList() {
                int tempValue = (int) getTempValue() + offset;
                list.add(tempValue);
                if(list.size() > 20) {
                        try {l20 = list.subList(list.size()-20, list.size());}
                        catch (IndexOutOfBoundsException e)
                        {       e.printStackTrace();
                                l20 = list;
                        } catch (NullPointerException e) {
                                l20 = new ArrayList<Integer>();
                        }
                }
                else {l20 = list;}
                System.out.println(l20);
```

```
	}

	//sets the sample rate
	public int setSampleRate() {
		int sampleRate = gui.getSampleRate();
		while (sampleRate != 0) {
			this.tempList();
			gui.Arraytemp(120);
			try {
				Thread.sleep(sampleRate);
			}
			catch (InterruptedException e) {
				e.printStackTrace();
			}
			//updates the sample rate value
			if(sampleRate!= gui.getSampleRate()) {
				sampleRate = gui.getSampleRate();

			}
		}
		System.out.println(sampleRate);
		return sampleRate;
	}

	//specifically gets just temperature value, gets the TempService class
	//from ThreadedConnectionHandler in the server
	public double getTempValue() {
		String theTempValueCommand = "GetTemp";
		String theTempValue = "";
		this.send(theTempValueCommand);
		try{
			theTempValue = (String) receive();
		}
		catch (Exception e){
			System.out.println(e);
		}
		double tempvalue = Double.parseDouble(theTempValue)/1000;
		return tempvalue;

	}
	 public void LEDon() {
		String ledoncommand = "LEDON", ledon;
		System.out.println("01. -> Sending Command (" + ledoncommand + ")
to the server...");
		this.send(ledoncommand);
		try{

			ledon = (String) receive();
			System.out.println("05. <- The Server responded with: ");
			System.out.println("    <- " + ledon);

		}
		catch (Exception e){
			System.out.println("XX. There was an invalid object sent
back from the server");
		}
		System.out.println("06. -- Disconnected from Server.");

	    }
```

```
        public void LEDoff() {
              String ledoffcommand = "LEDOFF", ledoff;
              System.out.println("01. -> Sending Command (" + ledoffcommand + ")
to the server...");
              this.send(ledoffcommand);
              try{

                    ledoff = (String) receive();
                    System.out.println("05. <- The Server responded with: ");
                    System.out.println("    <- " + ledoff);

              }
              catch (Exception e){
                    System.out.println("XX. There was an invalid object sent
back from the server");
              }
              System.out.println("06. -- Disconnected from Server.");

           }
        public void getDate() {
              String theDateCommand = "GetDate", theDateAndTime;
              System.out.println("01. -> Sending Command (" + theDateCommand +
") to the server...");
              this.send(theDateCommand);
              try{
                    String timeStamp = new SimpleDateFormat("yyyy/MM/dd
HH.mm.ss").format(new java.util.Date());
                    theDateAndTime = " "+ timeStamp +receive();
                    System.out.println("05. <- The Server responded with: ");
                    System.out.println("    <- " + theDateAndTime);

              }
              catch (Exception e){
                    System.out.println("XX. There was an invalid object sent
back from the server");
              }
              System.out.println("06. -- Disconnected from Server.");

           }

    private void send(Object o) {
              try {
                        System.out.println("02. -> Sending an object...");
                        os.writeObject(o);
                        os.flush();
              }
           catch (Exception e) {
                System.out.println("XX. Exception Occurred on Sending:" +
e.toString());
              }
    }

    // method to receive a generic object.
    private Object receive()
    {
              Object o = null;


              try {
```

```
                            System.out.println("03. -- About to receive an
object...");
                        o= is.readObject();
                        System.out.println("04. <- Object received...");
                        }
           catch (Exception e) {
                 System.out.println("XX. Exception Occurred on Receiving:" +
e.toString());
              }

              return o;
    }


}
```