CS 2310 FINAL REPORT

PERSONAL HEALTHCARE SYSTEM

AMEYA DAPHALAPURKAR

- **INTRODUCTION:**

After learning the SIS test bed system, I decided that I would like to implement a Patient's Pulse Rate Tracker. The sensor used to note the readings in this project is 'CMS 50D+ Blue Finger Pulse Oximeter'. It returns readings of patient's pulse rates continuously which is monitored by the system. Using sensors eases the patient monitoring wherever the system is prevalent. Also, there are provisions made to monitor and store this output from the sensors in files that are globally accepted formats like JSON and XML so that we can plot the graphs of readings of patient and can do any analysis as required.

The components involved in the setup are:

1. Input processor
2. Health Monitor
3. Uploader
4. Health Care Staff

Input processor takes the reading from the sensor and forwards it to the health Monitor so that it can analyze the condition the patient is currently in. Input processor has the job of filtering in and out garbage values as the censor spits out many values at random and the trick part is in getting the right value from the sensor.

Monitor Component analyzes the state of the patient using the readings that the sensor is currently projecting using the input processor. The heart rate is the number of times per minute that the heart beats. Heart rate rises significantly in response to adrenaline if a patient under monitoring is frightened or surprised and thus taking the patient's pulse is a direct measure of the heart rate. A normal adult resting heart beat is between 60-100 heartbeats per minute whereas for some athletes it can also lower down till 50.

This project's monitoring system can tell us if the patient has Tachycardia or Bradycardia. Tachycardia refers to the heart beating too fast, that is at rest over 100 beats per minute. Bradycardia refers to the heart beating too slow, which is usually below 60 beats per minute.

The patient's pulse rate can also be calculated while he or she is sleeping, remotely by the doctor. This also helps the patient to avoid any costs of visiting the hospital center for any diagnostic regular checkups and helps the doctor to view the statistical data all at once by checking the readings.

Uploader has the important function of uploading the data on to the internet. This is done by using two methods, email, and database entry. The chronobot database is accessed and an entry is made into the database.

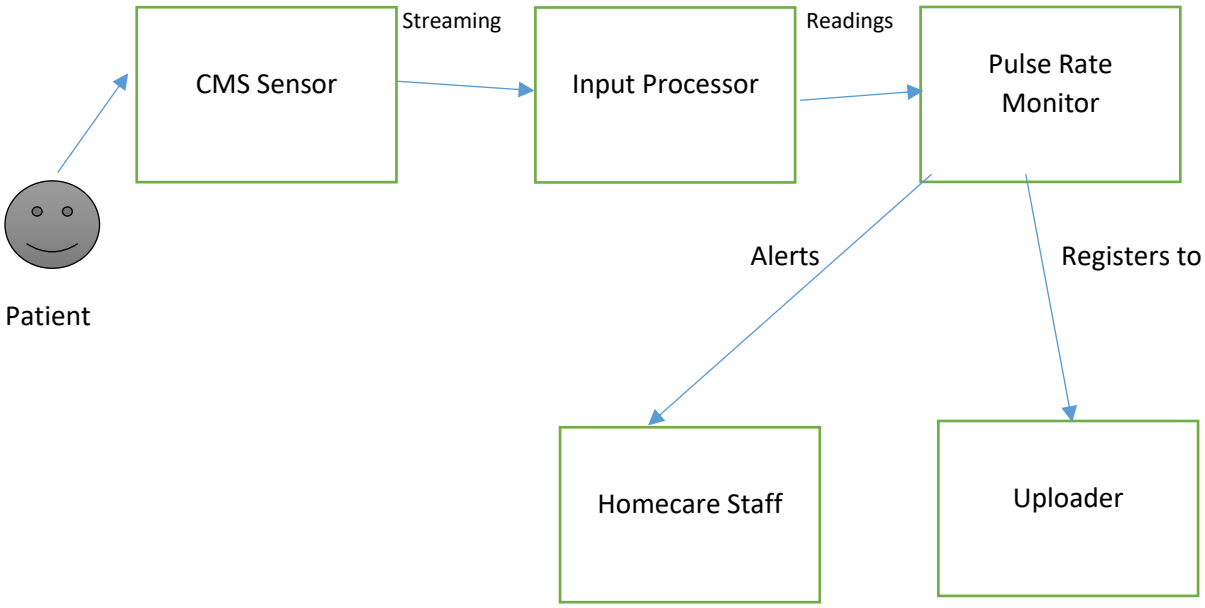
If the reading is an abnormal reading, the Health Care Staff component is triggered and a GUI display is generated where it displays that "A patient visit protocol is initialized."

- **Readings and their Meanings:**

  - Below 60: Bradycardia
  - Between 60 and 100: Normal
  - Above 100: Tachycardia

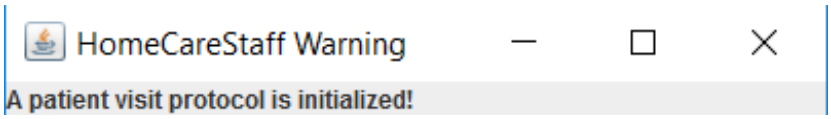
- **Scenario:**

The scenario is considered like the Exercise 4 but now here there is a real Pulse Rate sensor instead of a Gesture Recognizer component. The patient's pulse rate is detected in real time and noted by the monitor as either normal or tachycardia or bradycardic. The HealthCare Staff manager system will detect this. Uploader at the same time notes the readings in to the database.



- **Screens of the Components:**

1. Warning from Health Care Staff:



2. Input Processor

Select InputProcessor

```
Should send to reading : 127
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
Should send to reading : 79
```

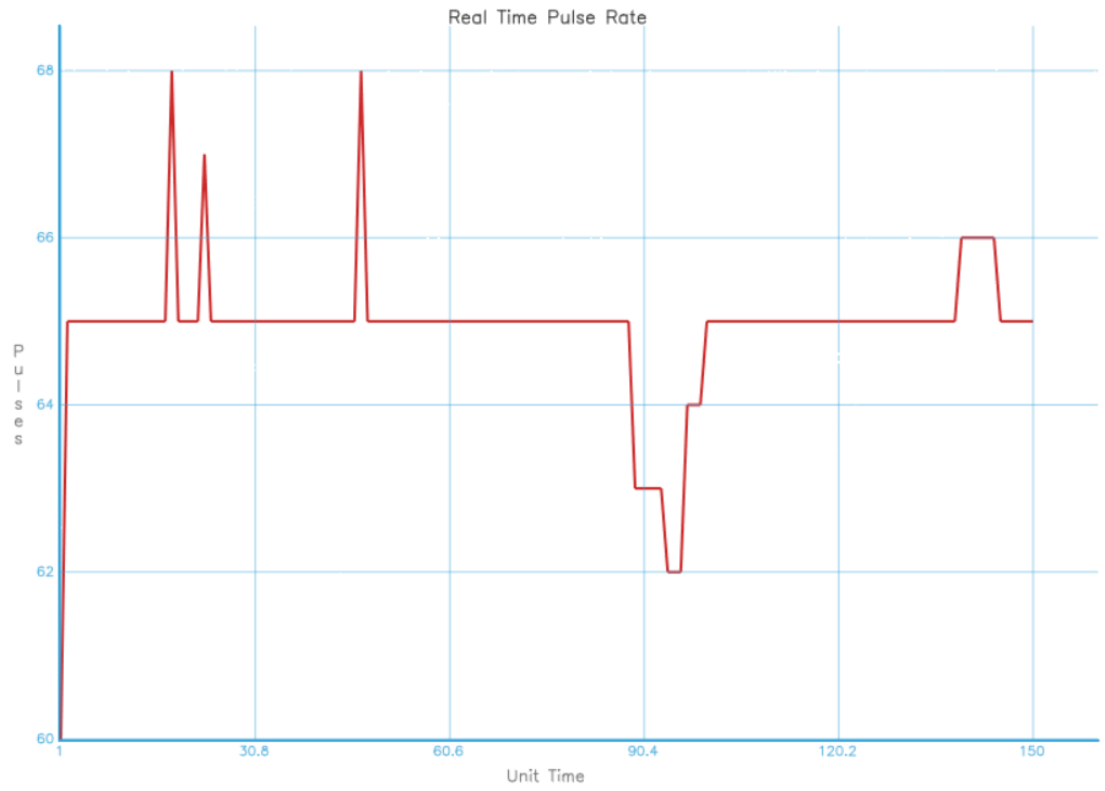
3. Health Monitor

HealthMonitor

```
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
Pulse rate: 79 beats per minute at 08/12/2016 01:11:05: Normal
```

Extra Deeds:

1. Visualization of Sensor Data using JSCharts:



For extra deed, I have implemented a visualization of the data from the sensor. The program starts writing on a file using xml code. The data values for the graph keep getting added for every set of 150 readings. So, this graph can be dynamically updated after every 150 readings.

2. Implemented functionality to format the data in JSON format, this not only makes it a globally compatible format for any data analysis or visualization but also makes it easily readable.

3. The Health Care Staff uses graphical UI to alert with its warning messages adding a bit of liveliness to the command line setup.

- **APPENDIX**:

1. SimpleRead.java uses java libraries such as comm to understand the readings from the sensor.

```
import java.io.*;
import java.net.ConnectException;
import java.net.Socket;
import java.util.*;
import javax.comm.*;



public class SimpleRead implements Runnable, SerialPortEventListener {
	static CommPortIdentifier portId;
	static Enumeration portList;
	int handle = 0;
	static int skipper = 0;
	InputStream inputStream;
	SerialPort serialPort;
	Thread readThread;
	static PrintWriter pulseWriter;
	public static void main(String[] args) throws IOException {
		pulseWriter = new PrintWriter(new BufferedWriter(new FileWriter("breathReadings2.xml", true)));
		pulseWriter.println("<?xml version=\"1.0\"?>");
		pulseWriter.println("<JSChart>");
		pulseWriter.println("<dataset type=\"line\">");
		//pulseWriter.println("<data unit=\"" + skipper + "\" value=\"" + pulse + "\"/>");

		portList = CommPortIdentifier.getPortIdentifiers();
		while (portList.hasMoreElements()) {
			portId = (CommPortIdentifier) portList.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				if (portId.getName().equals("COM3")) {
					SimpleRead reader = new SimpleRead();
				}
			}
		}
	}

	public SimpleRead() {
		try {
			serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
		} catch (PortInUseException e) {
			System.out.println(e);
		}
		try {
			inputStream = serialPort.getInputStream();
		} catch (IOException e) {
			System.out.println(e);
		}
		try {
			serialPort.addEventListener(this);
		} catch (TooManyListenersException e) {
			System.out.println(e);
		}
		serialPort.notifyOnDataAvailable(true);
		try {
```

```
                    serialPort.setSerialPortParams(19200, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);
                } catch (UnsupportedCommOperationException e) {
                    System.out.println(e);
                }
                readThread = new Thread(this);
                readThread.start();
        }

        public void run() {
                try {
                    Thread.sleep(20000);
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
        }

        public void serialEvent(SerialPortEvent event) {

                switch (event.getEventType()) {
                case SerialPortEvent.BI:
                case SerialPortEvent.OE:
                case SerialPortEvent.FE:
                case SerialPortEvent.PE:
                case SerialPortEvent.CD:
                case SerialPortEvent.CTS:
                case SerialPortEvent.DSR:
                case SerialPortEvent.RI:
                case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                    break;
                case SerialPortEvent.DATA_AVAILABLE:
                    handle++;
                    if(skipper==150){

                            pulseWriter.println("</dataset>");
                            pulseWriter.println("<optionset>");
                            /*pulseWriter.println("<option set=\"setLineColor\" value=\"'#8D9386'\"/>");
                            pulseWriter.println("<option set=\"setLineWidth\" value=\"4\"/>");
                            pulseWriter.println("<option set=\"setTitleColor\" value=\"'#7D7D7D'\"/>");
                            pulseWriter.println("<option set=\"setAxisColor\" value=\"'#9F0505'\"/>");
                            pulseWriter.println("<option set=\"setGridColor\" value=\"'#a4a4a4'\"/>");
                            pulseWriter.println("<option set=\"setAxisValuesColor\" value=\"'#333639'\"/>");
                            pulseWriter.println("<option set=\"setAxisNameColor\" value=\"'#333639'\"/>");
                            pulseWriter.println("<option set=\"setTextPaddingLeft\" value=\"0\"/>");
                                pulseWriter.println("</optionset>");
                                pulseWriter.println("</JSChart>");
                                pulseWriter.close();*/

                            pulseWriter.println("<option set=\"setAxisNameFontSize\" value=\"10\"/>");
                            pulseWriter.println("<option set=\"setAxisNameX\" value=\"'Unit Time'\"/>");
                            pulseWriter.println("<option set=\"setAxisNameY\" value=\"'Pulses'\"/>");
                            pulseWriter.println("<option set=\"setAxisNameColor\" value=\"'#787878'\"/>");
                            pulseWriter.println("<option set=\"setAxisValuesNumberX\" value=\"6\"/>");
                            pulseWriter.println("<option set=\"setAxisValuesNumberY\" value=\"5\"/>");
                            pulseWriter.println("<option set=\"setAxisValuesColor\" value=\"'#38a4d9'\"/>");
                            pulseWriter.println("<option set=\"setAxisColor\" value=\"'#38a4d9'\"/>");
                            pulseWriter.println("<option set=\"setLineColor\" value=\"'#C71112'\"/>");
                            pulseWriter.println("<option set=\"setTitle\" value=\"'Real Time Pulse Rate'\"/>");
```

```
                                        pulseWriter.println("<option set=\"setTitleColor\" value=\"'#383838'\"/>");
                                        pulseWriter.println("<option set=\"setGraphExtend\" value=\"true\"/>");
                                        pulseWriter.println("<option set=\"setGridColor\" value=\"'#38a4d9'\"/>");
                                        pulseWriter.println("<option set=\"setSize\" value=\"1000, 600\"/>");
                                        pulseWriter.println("<option set=\"setAxisPaddingLeft\" value=\"140\"/>");
                                        pulseWriter.println("<option set=\"setAxisPaddingRight\" value=\"140\"/>");
                                        pulseWriter.println("<option set=\"setAxisPaddingTop\" value=\"60\"/>");
                                        pulseWriter.println("<option set=\"setAxisPaddingBottom\" value=\"45\"/>");
                                        pulseWriter.println("<option set=\"setTextPaddingLeft\" value=\"105\"/>");
                                        pulseWriter.println("<option set=\"setTextPaddingBottom\" value=\"12\"/>");
                                        pulseWriter.println("<option set=\"setBackgroundImage\"
value=\"'path/background.jpg'\"/>");
                                        pulseWriter.println("</optionset>");
                                        pulseWriter.println("</JSChart>");
                                        pulseWriter.close();
                                                System.exit(0);
                                }
                                Socket socket = null;
                                        byte[] readBuffer = new byte[20];
                                        int ascii;

                                        try {
                                                while (inputStream.available() > 0) {

                                                        int numBytes = inputStream.read(readBuffer);
                                                }
                                                int index = 0;
                                                Byte regular = readBuffer[index];

                                                int pulse = (int)regular;

                                                if (pulse > 10){
                                                skipper++;
                                                pulseWriter.println("<data unit=\"" + skipper + "\" value=\"" + pulse + "\"/>");
                                                System.out.println("Pulse Rate :" + pulse);
                                                try {
                                            socket = new Socket("127.0.0.1",54000);
                                            DataOutputStream d = new DataOutputStream(socket.getOutputStream());
                                            d.writeUTF(pulse + "");
                                            } catch (ConnectException e) {
                                                e.printStackTrace();
                                            }
                                                }
                                        } catch (IOException e) {
                                                System.out.println(e);
                                        }

                                break;
                        }
                }
}
```

2. Input Processor takes the reading and forwards them to monitor

```
import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.Date;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Scanner;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.net.*;

public class FakeBreathingRate {
	// socket for connection to SISServer
	static Socket universal;
	// message writer
	static MsgEncoder encoder;
	// message reader
	static MsgDecoder decoder;
   static ServerSocket s;
	// scope of this component
	private static final String SCOPE = "SIS.Scope1";
	// name of this component
   private static final String NAME = "FakeBreathRate";
	// messages types that can be handled by this component
	private static final List<String> TYPES = new ArrayList<String>(
				Arrays.asList(new String[] { "Setting", "Confirm" }));

	private static int refreshRate = 500, max, min = 90;
	private static Date startDate = new Date(), endDate = new Date();

	private static Timer timer = new Timer();

	// shared by all kinds of records that can be generated by this component
	private static KeyValueList record = new KeyValueList();
	// shared by all kinds of alerts that can be generated by this component
	private static KeyValueList alert = new KeyValueList();

	private static SPO2Reading reading = new SPO2Reading();
	static Scanner p = new Scanner(System.in);
	static int threshold = 0;
	/*
	 * Main program
	 */
	public static void main(String[] args) throws IOException{
		try {
			// try to establish a connection to SISServer
			universal = connect();

			// bind the message reader to inputstream of the socket
			decoder = new MsgDecoder(universal.getInputStream());
			// bind the message writer to outputstream of the socket
			encoder = new MsgEncoder(universal.getOutputStream());

			/*
			 * construct a Connect message to establish the connection
			 */
			KeyValueList reg = new KeyValueList();
			reg.putPair("Scope", SCOPE);
```

```
                        reg.putPair("MessageType", "Register");
                        reg.putPair("Role", "Basic");
                        reg.putPair("Name", NAME);

                        encoder.sendMsg(reg);

                        KeyValueList conn = new KeyValueList();
                        conn.putPair("Scope", SCOPE);
                        conn.putPair("MessageType", "Connect");
                        conn.putPair("Role", "Basic");
                        conn.putPair("Name", NAME);
                        encoder.sendMsg(conn);
                        // KeyValueList for inward messages, see KeyValueList for
                        // details
                        KeyValueList kvList;
                        DateFormat dateFormat = new SimpleDateFormat(("dd/MM/yyyy HH:mm:ss"));

                        Scanner scanner = new Scanner(System.in);
s = new ServerSocket(54000);



                        while(true){
                                //System.out.print("Enter breathing rate to simulate: ");
                                //String reading = "80";

  Socket socketL;
  String str="";
                                try {
                                        socketL = s.accept();
    DataInputStream din = new DataInputStream(socketL.getInputStream());
    str = din.readUTF();
    System.out.println("Should send to reading : " + str);
    din.close();
    socketL.close();
  }
  catch (Exception ex1) {
                                                ex1.printStackTrace();
                                        }

  record.putPair("Scope", SCOPE);
                                record.putPair("MessageType", "Reading");
                                record.putPair("Sender", NAME);
                                record.putPair("Date", dateFormat.format(new Date()));
                                record.putPair("BreathRate", str);

                                encoder.sendMsg(record);
                        }



                } catch (Exception e) {
                        // if anything goes wrong, try to re-establish the connection
                        try {
                                // wait for 1 second to retry
                                Thread.sleep(1000);
                        } catch (InterruptedException e2) {
                        }
```

```
                                System.out.println("Try to reconnect");
                                try {
                                        universal = connect();
                                } catch (IOException e1) {
                                }
                        }
                }

                /*
                 * used for connect(reconnect) to SISServer
                 */
                static Socket connect() throws IOException {
                        Socket socket = new Socket("127.0.0.1", 7999);
                        return socket;
                }
}

class SPO2Reading {
        int spo2;
        long date;
}
```

3. Health Care Monitor is the main analyzing component

```
import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.io.*;
import org.json.simple.JSONObject;
import javax.swing.*;
import java.util.Scanner;

public class HealthMonitor {
        // socket for connection to SISServer
        private static Socket universal;
        // message writer
        private static MsgEncoder encoder;
        // message reader
        private static MsgDecoder decoder;
        // scope of this component
        private static final String SCOPE = "SIS.Scope1";
        // name of this component
   private static final String NAME = "HealthMonitor";
        // messages types that can be handled by this component

        private static final List<String> TYPES = new ArrayList<String>(
                        Arrays.asList(new String[] { "Setting", "Confirm" }));

        private static int refreshRate = 500, max = 140, min = 90;
        private static Date startDate = new Date(), endDate = new Date();

        private static Timer timer = new Timer();
```

```
	// shared by all kinds of records that can be generated by this component
	private static KeyValueList record = new KeyValueList();
	// shared by all kinds of alerts that can be generated by this component
	private static KeyValueList alert = new KeyValueList();

	private static SPO2Reading reading = new SPO2Reading();
	static Scanner p = new Scanner(System.in);
	static int threshold = 0;
	/*
	 * Main program.
	 */
	public static void main(String[] args) {
		try {
			// try to establish a connection to SISServer
			universal = connect();

			// bind the message reader to inputstream of the socket
			decoder = new MsgDecoder(universal.getInputStream());
			// bind the message writer to outputstream of the socket
			encoder = new MsgEncoder(universal.getOutputStream());

			/*
			 * construct a Connect message to establish the connection
			 */
			KeyValueList conn = new KeyValueList();
			conn.putPair("Scope", SCOPE);
			conn.putPair("MessageType", "Register");
			conn.putPair("Role", "Monitor");
			conn.putPair("Name", NAME);

			encoder.sendMsg(conn);

			KeyValueList connect = new KeyValueList();
			connect.putPair("Scope", SCOPE);
			connect.putPair("MessageType", "Connect");
			connect.putPair("Name", NAME);
			KeyValueList confirm = new KeyValueList();

			encoder.sendMsg(connect);

			confirm = decoder.getMsg();

			if (confirm.getValue("MessageType").equalsIgnoreCase("Confirm")){
				System.out.println("connected");
			}

			PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter("sp02readings.txt",
true)));
			PrintWriter breathWriter = new PrintWriter(new BufferedWriter(new
FileWriter("breathReadings.txt", true)));

			// componentTask();

			// KeyValueList for inward messages, see KeyValueList for
			// details
			KeyValueList kvList;

			System.out.println("Please enter your pulse rate threshold :");
```

```
                threshold = p.nextInt();

                int flagg = 0;
                boolean UFlag = false;
                while (true) {
                        int spo2Lvl = 0;
                        KeyValueList list = new KeyValueList();
                        list = decoder.getMsg();

                        String scope = list.getValue("Scope");
                        String msgType = list.getValue("MessageType");
                        String sender = list.getValue("Sender");
                        String date = list.getValue("Date");
                        String breathRate = "";
                        try{
                                if(sender.compareTo("FakeBreathRate") == 0){
                                        breathRate = list.getValue("BreathRate");
                                        if      (Integer.parseInt(breathRate) >= threshold &&
Integer.parseInt(breathRate) <= threshold + 10){
                                        System.out.println("Pulse rate: " + breathRate + " beats per minute
at " + date + ": Normal");
                                        }
                                        else if (Integer.parseInt(breathRate) > threshold + 10){
                                                System.out.println("Pulse rate: " + breathRate + " beats
per minute at " + date + ": Tachycardia");

                                                flagg += 1;
                                        }
                                        else if(Integer.parseInt(breathRate) < threshold){
                                                System.out.println("Pulse rate: " + breathRate + " beats
per minute at " + date + ": Bradycardia");

                                                flagg += 1;
                                        }
                                        //breathWriter.println("{\"Date\": \"" + date + "\",
\"BeatsPerMinute\": \"" + breathRate + "\"}");
                                        breathWriter.println(breathRate);
                        breathWriter.flush();

                                        if(flagg == 1)
                                        {
                                                flagg += 1;
                                                UFlag = true;
                                                Socket socket = null;
                                                socket = new Socket("127.0.0.1",55000);
                                                DataOutputStream d = new
DataOutputStream(socket.getOutputStream());
                                        d.writeUTF("emergency");
                                                d.close();
                                                socket.close();
                                        }
                                        if(UFlag){
                                                UFlag = false;
                                                Socket socket = null;
                                                socket = new Socket("127.0.0.1",53000);
                                                DataOutputStream d = new
DataOutputStream(socket.getOutputStream());
                                        d.writeUTF(breathRate);
                                                d.close();
                                                socket.close();
```

```
                                        }

                                        }

                                }
                                catch(Exception e){
                                        System.out.println(e);
                                }
                        }

                } catch (Exception e) {
                        // if anything goes wrong, try to re-establish the connection
                        e.printStackTrace();
                        try {
                                // wait for 1 second to retry
                                Thread.sleep(1000);
                        } catch (InterruptedException e2) {
                        }
                        System.out.println("Try to reconnect");
                        try {
                                universal = connect();
                        } catch (IOException e1) {
                        }
                }
        }



        /*
         * used for connect(reconnect) to SISServer
         */
        private static Socket connect() throws IOException {
                Socket socket = new Socket("127.0.0.1", 7999);
                return socket;
        }
}

class SPO2Reading{
        public int SPO2;
        public long date;
}
```