Sri Lanka Institute of Information Technology

Fire Alarm System

Report

Distributed Systems

Programming Assignment 1

Submitted by:

IT16030190 – V.A.Wickramasinghe

Table of Contents

High Level Software Architecture Diagram

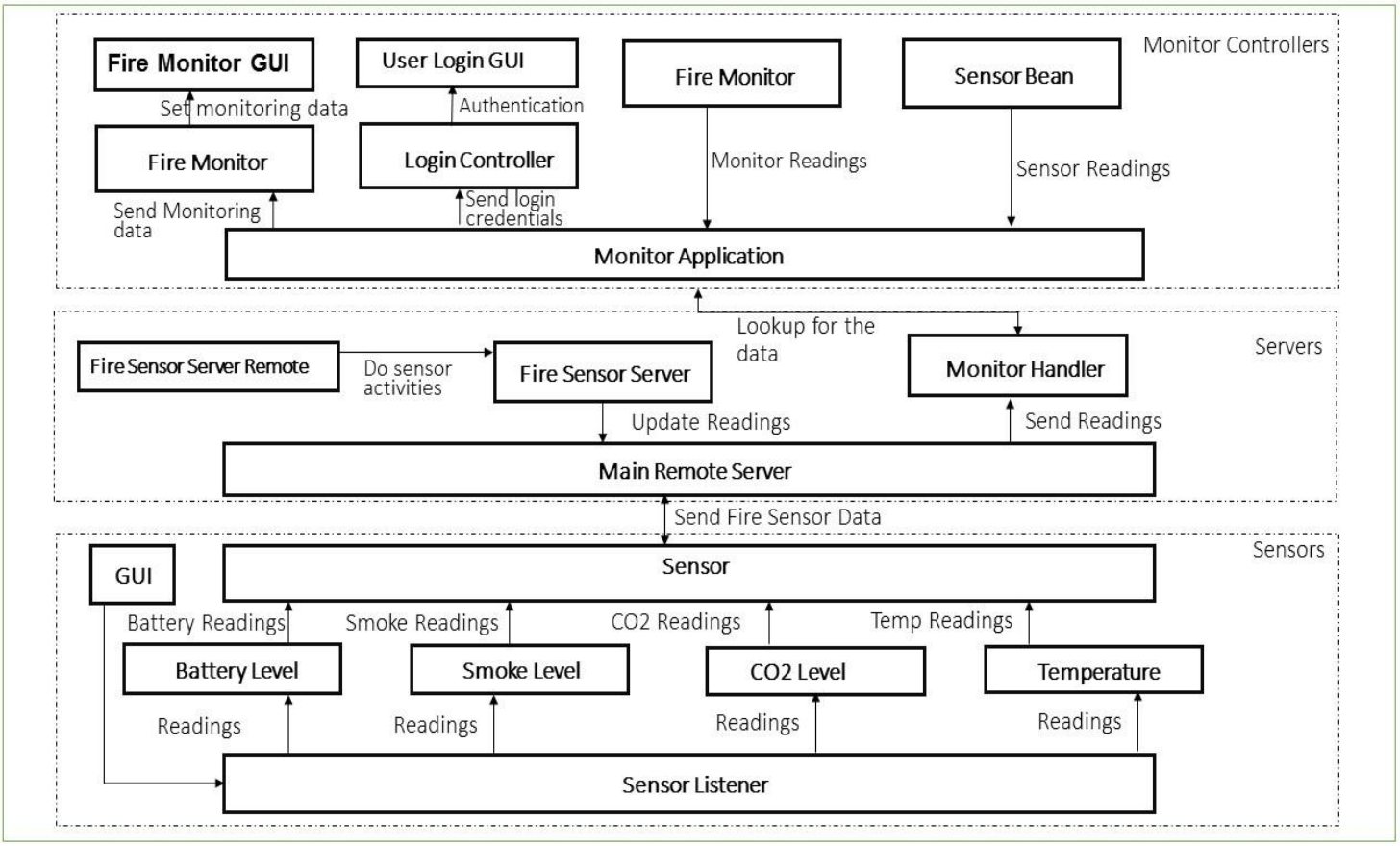

Figure 1

System Architecture

This Fire Alarm System is implemented using Java, RMI, Sockets and Asynchronous communication. In here Monitors and servers communicate each other using Java RMI while sensors and servers communicate each other using socket connection. There are 4 sensors and each reading is serialized and passed to the socket communication. Using RMI monitors able to invoke the remote methods in the servers. There are 2 instances of authentication. When client try to log on to the system and when admin tries to add new monitor to the system authentication is done. Fire Sensor and Login Controller will authenticate the logins. This system is more biased towards layered architecture where clients/monitors acquire sensor readings from the servers in several layers. There is a bidirectional communication between Sensor and Main Remote Server to get updated readings frequently and those readings will be sent to Monitor Handler. In here monitor handler, Sensor bean and fire monitor acquire readings and send to the monitor application. It do the necessary processing and send the readings to the required interfaces. This system deployed on main remote server.

Class Diagram

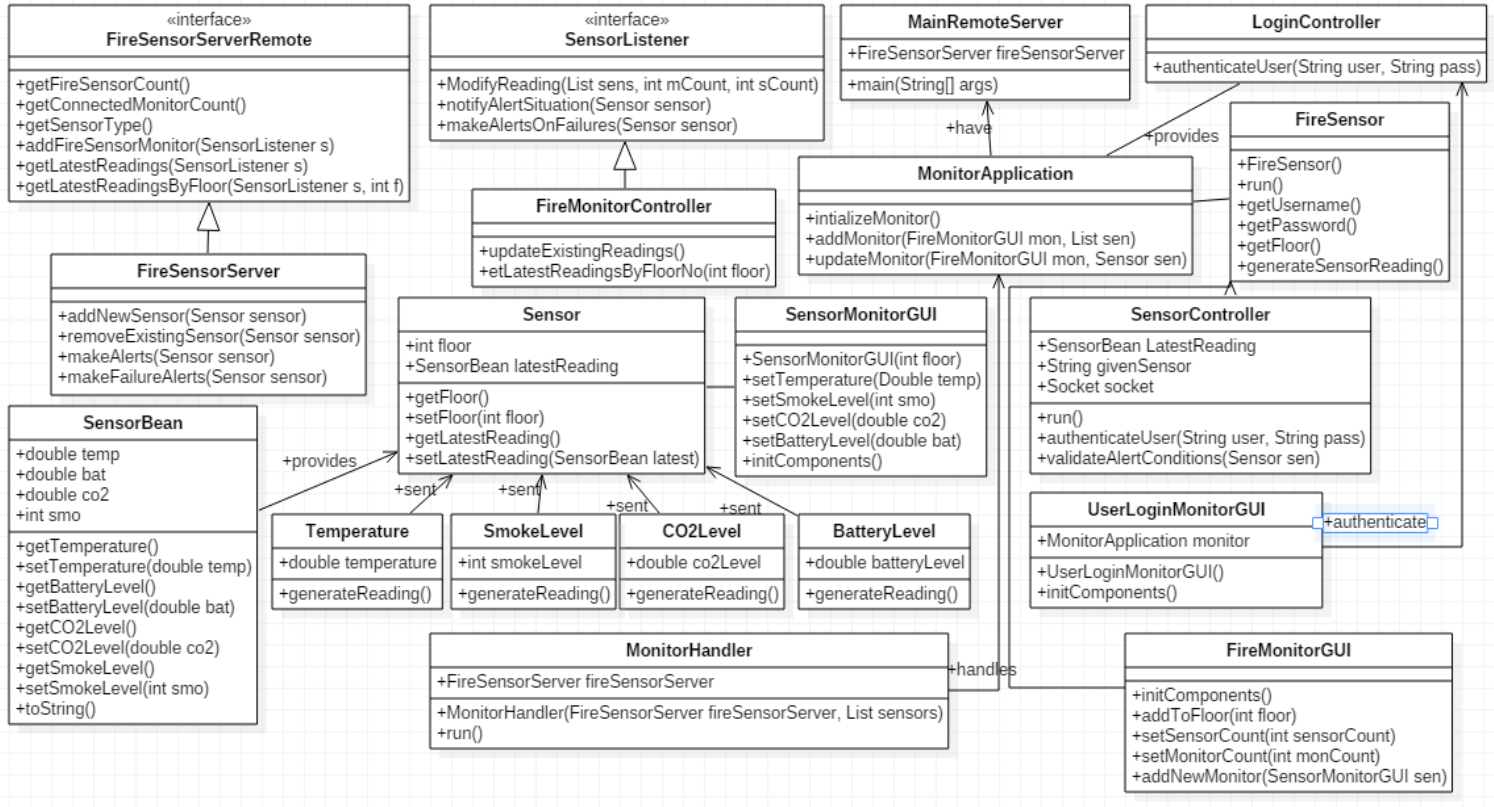

Figure 2

Sequence Diagrams

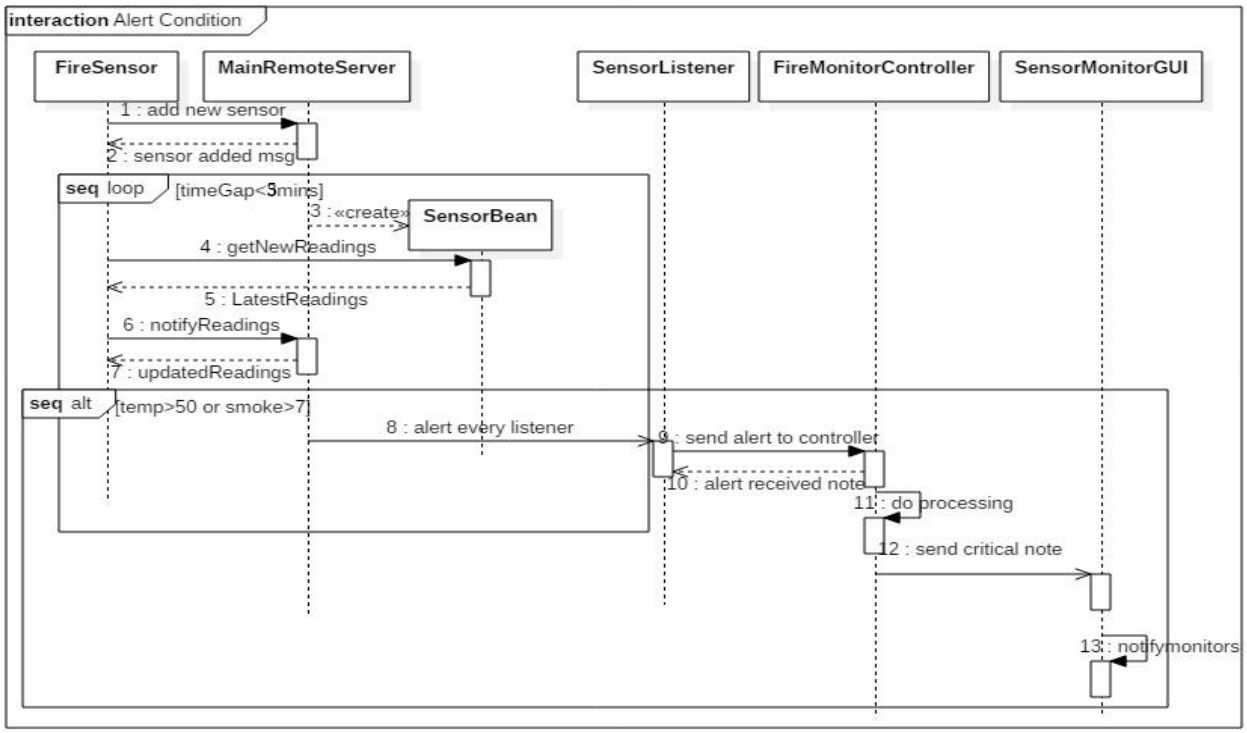

Figure 3

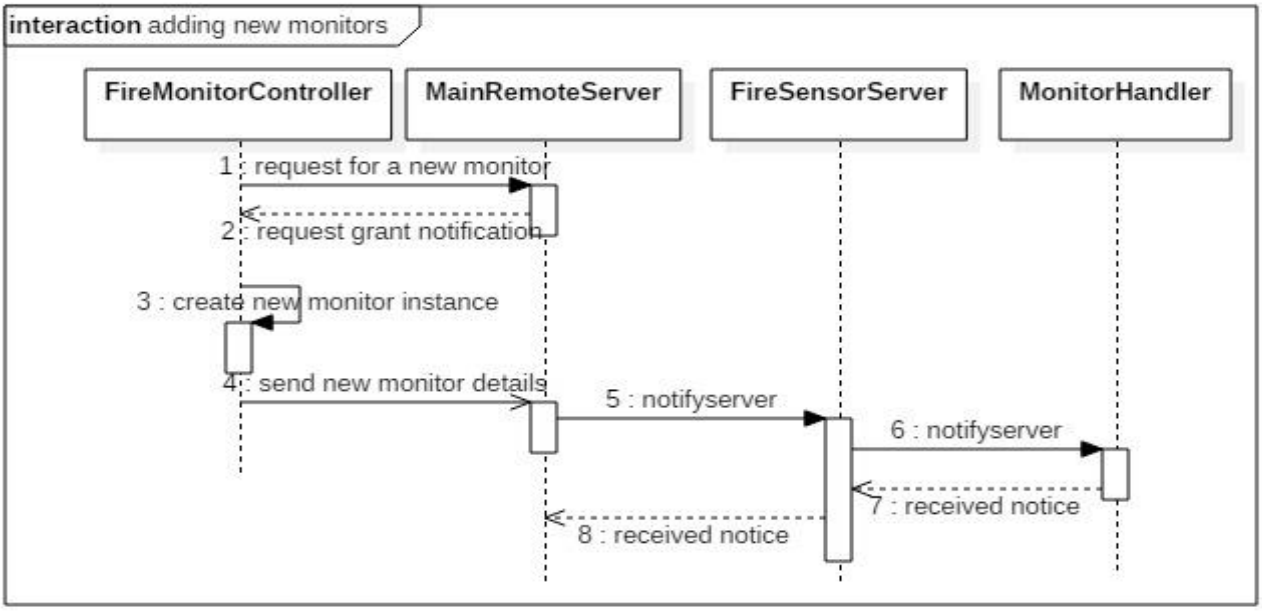

Figure 4

Communication between the Sensors and the Server

Sensors used in this fire alarm system is work on TCP/IP network. Therefore socket communication can be used to communicate with the servers. These sockets act as the endpoint between the sensors and servers.

```
   // Socket Connection is created
        String server = "127.0.0.1";
        Socket socket = new Socket(server, 9001);
```

In here server is defined as the localhost and new socket connection is created in between the server and sensors where as server works on the port 9001.

```
   import java.net.Socket;
```

Used to implement the client socket connection and uses input and output streams in passing the data in between server and sensors.

```
//get user input and output
  bufferedReader = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
```

Buffered reader is used to get the sensor data using the socket connection as an input stream

```
 printWriter = new PrintWriter(socket.getOutputStream(), true);
```

Printwriter is used to output the sensor data to the server using the socket connection created.

```
public SensorController(Socket socket, FireSensorServer
fireSensorServer) {
       this.socket = socket;
       this.sensor = new Sensor();
       this.fireSensorServer = fireSensorServer; }
```

In here server and sensors are set using the constructor for the server communication to be established.

```
import java.net.ServerSocket;
```

Using this import server Socket communication is established.It waits until the sensors send data to it to initiate the communication.

```
// Initiate Server Socket Connection
        try (ServerSocket serverSocket = new ServerSocket(9001)) {
            while (true) {
                SensorController sensorHandler = new
SensorController(serverSocket.accept(),fireSensorServer);
                sensorHandler.start();}
            }
```

In here using the accept method server read and write data to and from the sockets connected to it. Finally the close() is called to close the connection

```
//close the connection
                  socket.close();
```

Communication between the Monitors and the Server

In here servers and monitors communicate each other using the Remote Method Invocation. Asynchronous communication is established using call back methods as below.

```
public interface FireSensorServerRemote extends Remote{
   //callback methods
 //count no of connected sensors
 public int getFireSensorCount() throws RemoteException;

//count no of connected monitors
 public int getConnectedMonitorCount() throws RemoteException;
//code
}
```

Above interface is defined to be extend from the remote and throws remote exceptions which uses basics of RMI. Below imports are used for this.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

Server is bind with server which will be lookup to obtain the service.

```
// Bind to rmi registry and lookup for name
Registry reg = LocateRegistry.createRegistry(1099);
fireSensorServer = new FireSensorServer();
Naming.rebind("//127.0.0.1/fireSensor", fireSensorServer);
```

In here Naming.lookup is used to get the reference to the remote server

```
// lookup for remote object
FireSensorServerRemote fireSensorServerRemote =
(FireSensorServerRemote)
Naming.lookup("rmi://127.0.0.1/fireSensor");
```

Using the reference in here we call the local object of the method using RMI.

```
// monitors are added to remote object
fireSensorServerRemote.addFireSensorMonitor(monitor);

// Deserializing the records
try(ObjectInputStream objc = new ObjectInputStream
                             (new
ByteArrayInputStream(Base64.getDecoder().decode(input)))){

//code
 }
```

Objects are serialized and De serialized to a byte stream. Those data are sent to the GUI for displaying it's information. Serializable interfaces are used for this.

Assumptions

- New Servers and sensors can be put up by only an admin with rights.
- Only one person can access the sensors at one time. Simultaneous access is blocked to avoid threads accessing same data at once, which avoid inconsistent data.
- IOT sensors behave as programmed using java.
- Every floor there are four types of sensors namely temperature, smoke, CO_2 and battery level sensors.
- Sensor readings are passed to the servers after the authentication.
- Server is configured to run on port 9001.

Appendix

FireSensorServer.java

```
/** This class handles as the server which handles the sensor and monitor
activities

public class FireSensorServer extends UnicastRemoteObject implements
FireSensorServerRemote{

    private static final long serialVersionUID = 1L;
    //Fire Sensors
    private List<Sensor> fireSensor= new ArrayList<>();
    //FireSensorMonitors Connected with the RemoteServer
    private List<SensorListener> fireMonitor = new ArrayList<>();

    public FireSensorServer() throws RemoteException{}

    //count no of monitors connected
    @Override
    public int getConnectedMonitorCount() throws RemoteException{
        return fireMonitor.size();
    }

    //count no of sensors connected
    @Override
    public int getFireSensorCount() throws RemoteException{
        return fireSensor.size();
    }

    //get the sensor name
    @Override
    public List<Sensor> getSensorType() throws RemoteException {
        return fireSensor;
    }

    //getLatest readings
    @Override
    public void getLatestReadings(SensorListener sensorListener) throws
RemoteException  {
            sensorListener.ModifyReading(fireSensor,
getConnectedMonitorCount(), getFireSensorCount());
    }

    //add listeners
    @Override
    public void addFireSensorMonitor(SensorListener sensorListener)
throws RemoteException {
        System.out.println(sensorListener + " is added");
        //other threads cannot interfere while adding sensors
        synchronized(fireMonitor){
            fireMonitor.add(sensorListener);
        }
        // Start a new Thread for this monitor
        new MonitorHandler(this,fireSensor).start();
```

```
    //getLatest readings according to the floor defined
     @Override
     public Sensor getLatestReadingsByFloor(SensorListener
sensorListener, int floor) throws Exception {
         for(Sensor sen : fireSensor){
             if (sen.getFloor() == floor ){
                 return sen;
             }
         }

         throw new Exception();
     }

    //add new sensors
     public void addNewSensor(Sensor sensor) {
       System.out.println(sensor + " is added");
       //threadsafe
         synchronized(fireSensor){
             this.fireSensor.add(sensor);
         }
     }

    //remove existing sensors
     public void removeExistingSensor(Sensor sensor) {
       System.out.println(sensor + " is removed");
       //threadsafe
         synchronized(fireSensor){
             this.fireSensor.remove(sensor);
         }
     }

    //providing alerts to monitors
     public void makeAlerts(Sensor sensor){
         if(!fireMonitor.isEmpty()){
             // Notify all the listeners
             fireMonitor.forEach((listener) -> {
                 try {
                     System.out.println(sensor + " in danger");
                     // alerting the monitor
                     listener.notifyAlertSituation(sensor);
                 }
                 catch (ConnectException e){
                     // remove listeners if failure occurs
                   fireMonitor.remove(listener);
                 }
             });
         }
     }
     //hourly basis notification
     public void notifyListeners(List<Sensor> sensors) {
         if(!fireMonitor.isEmpty()){
             // Notify all the listeners
             fireMonitor.forEach((listener) -> {
                 try {
                     //readings get updated
                     listener.ModifyReading(sensors,
getConnectedMonitorCount(), getFireSensorCount());
```

```
                } catch (ConnectException e){
                  fireMonitor.remove(listener);
                    System.err.println("Error is  " + e);
                }
            });
        }
    }
    //providing alerts to monitors if sensor fails
    public void makeFailureAlerts(Sensor sensor){
        if(!fireMonitor.isEmpty()){
            // Notify all the listeners
            fireMonitor.for((listener) -> {
                try {
                  System.out.println(sensor + " is failed");
                    // notifying sensors
                    listener.makeAlertsOnFailures(sensor);
                }
                catch (ConnectException e){
                  // remove listeners if failure occurs
                  fireMonitor.remove(listener);
                }

            });
        }
    }


   }
```

FireSensorServerRemote.java

```
/** This class contains all the RMI call back methods

public interface FireSensorServerRemote extends Remote{

   //count no of connected sensors
    public int getFireSensorCount() throws RemoteException;

   //count no of connected monitors
    public int getConnectedMonitorCount() throws RemoteException;

   //add new sensor Listener
    public void addFireSensorMonitor(SensorListener sensorListener)
throws RemoteException;

   //get sensor readings
    public void getLatestReadings(SensorListener sensorListener) throws
RemoteException ;


  //get readings according to given floor
    public Sensor getLatestReadingsByFloor(SensorListener
sensorListener, int floor) throws Exception ;

  //get sensors list
    public List<Sensor> getSensorType() throws RemoteException;

   }
```

MainRemoteServer.java

```
/**Main server which creates the rmi registry and looksup for the service

public class MainRemoteServer {

    // Server Listening port
    private static final int SERVERPORT = 9001;
    // Injecting Remote service
    private static FireSensorServer fireSensorServer;


    public static void main(String[] args) throws Exception {
        System.out.println("Finding a server......");

        try{
            // Bind to RMI registry and lookup for name
            fireSensorServer = new FireSensorServer();
            FireSensor fireSensor = new FireSensor();
            Registry reg = LocateRegistry.createRegistry(1099);

            Naming.rebind("//127.0.0.1/fireSensor", fireSensorServer);
            //sleep for 0.5s
            Thread.sleep(500);
            System.out.println("Server is Connected.");
        }


        // Initiate Server Socket Connection
        try (ServerSocket serverSocket = new ServerSocket(SERVERPORT))
{
            while (true) {
                SensorController sensorController = new
SensorController(serverSocket.accept(),fireSensorServer);
                sensorController.start();
            }
        }


    }
}
```

MonitorHandler.java

```
/** This class handles all the clients who are connected with the server

public class MonitorHandler extends Thread implements Runnable {

        private FireSensorServer fireSensorServer;
        private List<Sensor> sensors;

        //Set constructors values
        public MonitorHandler(FireSensorServer fireSensorServer,
List<Sensor> sensors) {
            this.fireSensorServer = fireSensorServer;
            this.sensors = sensors;
        }
```

```
        //override run method in thread class
        @Override
        public void run(){

            while(true){
                  // Every 1 hour listeners get notified
                try {
                    Thread.sleep(3600000);

                    fireSensorServer.notifyListeners(sensors);

                    System.out.println("Notification is done
Successfully!");
                }
            }
        }
    }
```

FireMonitorController.java

```
/**Act as the middleTier which which transfer readings with server and
client.Implements all callback methods

public class FireMonitorController extends UnicastRemoteObject
implements SensorListener {

    private MonitorApplication monitorApplication;
    private FireSensorServerRemote fireSensorRemoteServer;
    private FireMonitorController fireMonitor;
    private Sensor sensor;
    private FireMonitorGUI fireMonitorGUI;


    public FireMonitorController(FireSensorServerRemote sensor) throws
RemoteException{
        this.fireSensorRemoteServer = sensor;
        this.monitorApplication = new MonitorApplication();
    }

    //display the monitor controller GUI to user
    public void displayMonitor(FireMonitorController fireMon) throws
RemoteException{
      fireMonitor = fireMon;
        // create a new instance of monitor
      fireMonitorGUI = new FireMonitorGUI(fireMon);
      //assign it's default property on closing
      fireMonitorGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      //display gui
      fireMonitorGUI.setVisible(true);

      //setting up relevant data

      fireMonitorGUI.setMonitorCount(fireSensorRemoteServer.getConnecte
dMonitorCount());
```

```
      fireMonitorGUI.setSensorCount(fireSensorRemoteServer.getFireSenso
rCount());
      fireMon.monitorController.addMonitor(fireMonitorGUI,
fireMon.fireSensorRemoteServer.getSensorType());

    }
    //retrieve all the latest readings
    public void updateExistingReadings(){
        try {
            fireSensorRemoteServer.getLatestReadings(fireMonitor);
        }
    }
    @Override
    public void makeAlertsOnFailures(Sensor sensor) throws
RemoteException {
        //update each sensor when sensors are malfunctioning
        JOptionPane.showMessageDialog(fireMonitorGUI,"Floor
"+sensor.getFloor()+" Sensors are malfunctioning","Alert
Situation",JOptionPane.ERROR_MESSAGE);
    }

    @Override
    public void ModifyReading(List<Sensor> sensors, int monitorCount,
int sensorCount) throws RemoteException {
       //Setting up values in interface
      fireMonitorGUI.setMonitorCount(monitorCount);
      fireMonitorGUI.setSensorCount(sensorCount);

      //Add monitors if they exists to controller
        if( sensors != null || !sensors.isEmpty() ){
            this.monitorController.addMonitor(fireMonitorGUI, sensors);

        }
        else{
            JOptionPane.showMessageDialog(null,"alert" ,"Please restart
your Server, failed to get Updates",JOptionPane.WARNING_MESSAGE);
        }
    }

    //retrieve all the latest readings according to the floor
    public void getLatestReadingsByFloorNo(int floor){
        try {
            Sensor sensor =
fireSensorRemoteServer.getLatestReadingsByFloor(fireMonitor,floor);
            //update monitors with sensors in each floor
            monitorController.updateMonitor(fireMonitorGUI, sensor);
        }
    }

    @Override
    public void notifyAlertSituation(Sensor sensor) throws
RemoteException {
      //update each sensor when new readings comes
      updateExistingReadings();
```

```
        JOptionPane.showMessageDialog(fireMonitorGUI,"Floor
"+sensor.getFloor()+" is in danger.","Alert
Situation",JOptionPane.WARNING_MESSAGE);
    }

}
```

MonitorApplication.java

```
/** This class initialize the components to run the fire monitors

public class MonitorApplication {

    private Map<Integer,SensorMonitorGUI> sensorMonitors = new
HashMap<>();
    private List<Integer> sensorLocations = new ArrayList<>();

    //start the monitor after initializing components
    public void intializeMonitor(){
        try {

            // lookup for remote object
            FireSensorServerRemote fireSensorServerRemote =
(FireSensorServerRemote)
                        Naming.lookup("rmi://127.0.0.1/fireSensor");

            // Instantiating fire monitor controller
            FireMonitorController monitor = new
FireMonitorController(fireSensorServerRemote);

            // monitors are added to remote object
            fireSensorServerRemote.addFireSensorMonitor(monitor);

            // display each monitor on GUI
            monitor.displayMonitor(monitor);
        }

    }

    public void addMonitor(FireMonitorGUI monitorGUI,List<Sensor>
sensors){

        sensors.forEach(sensor ->{
            SensorMonitorGUI SensorMonitor;
            //get floor number
            int floor = sensor.getFloor();

            // check whether floor no already exists
            if(!sensorMonitors.contains(floor)){
                SensorMonitor = new SensorMonitorGUI(floor);

                //floors are put to a list with their no
                sensorMonitors.put(floor, SensorMonitor);

                //New monitors are added to each floor
                monitorGUI.addNewMonitor(SensorMonitor);
```

```
                    //set floor numbers
                    monitorGUI.addToFloor(floor);

                    // Add sensors to each floor
                    sensorLocations.add(floor);
                }
                //when floor no already exists
                else{
                    monitor = sensorMonitors.get(sensor.getFloor());
                }

                //set the reading of each floor

monitor.setTemperature(sensor.getLatestReading().getTemperature());

monitor.setBatteryLevel(sensor.getLatestReading().getBatteryLevel());

monitor.setSmokeLevel(sensor.getLatestReading().getSmokeLevel());

monitor.setCO2Level(sensor.getLatestReading().getCO2Level());
        });

    }

    //update the monitor with data
    public void updateMonitor(FireMonitorGUI monitorGUI, Sensor
sensor){
      //check the floor no
        if(sensorMonitors.containsKey(sensor.getFloor())){
            SensorMonitorGUI monitor =
sensorMonitors.get(sensor.getFloor());
            //set readings to each sensor

monitor.setTemperature(sensor.getLatestReading().getTemperature());

monitor.setBatteryLevel(sensor.getLatestReading().getBatteryLevel());

monitor.setSmokeLevel(sensor.getLatestReading().getSmokeLevel());

monitor.setCO2Level(sensor.getLatestReading().getCO2Level());
        }
    }
}
```

SensorBean.java

```
/**This class holds all setters and getters of the sensors

public class SensorBean implements Serializable {

    private static final long serialVersionUID = 1L;
    private double temperature;
    private double batteryLevel;
    private double co2Level;
    private int smokeLevel;

    public double getTemperature() {
```

```
        return temperature;
    }

    public void setTemperature(double temperature) {
        this.temperature = temperature;
    }

    public double getBatteryLevel() {
        return batteryLevel;
    }

    public void setBatteryLevel(double batteryLevel) {
        this.batteryLevel = batteryLevel;
    }

    public double getCO2Level() {
        return co2Level;
    }

    public void setCO2Level(double co2Level) {
        this.co2Level = co2Level;
    }

    public int getSmokeLevel() {
        return smokeLevel;
    }

    public void setSmokeLevel(int smokeLevel) {
        this.smokeLevel = smokeLevel;
    }

   //Display current readings
    @Override
    public String toString() {
        return "Current Sensor Readings \n"
            + "Temperature is: " + temperature + "\n"
            + "Battery Level is: " + batteryLevel + "\n"
            + "Smoke Level is: " + smokeLevel + "\n"
            + "CO2 Level is: " + co2Level;
    }}
```

SensorController.java

```
/** This class handles all the activities related with the sensors

public class SensorController extends Thread implements Runnable {

    private SensorBean LatestReading;
    private String givenSensor;
    private final Socket socket;
    private final Sensor sensor;


    //Set is used to assign unique names for sensors
    private static HashSet<String> sensorTypes = new HashSet<String>();

    // Map sensors with unique identification given
    private static Map<String,Sensor> mappedSensors = new HashMap<>();
```

```
    // FireSensorServer class injected
      private FireSensorServer fireSensorServer;
      private String password;
      private int floor;
      private String input;

    public SensorController(Socket socket, FireSensorServer
fireSensorServer) {
       socket = socket;
       sensor = new Sensor();
       fireSensorServer = fireSensorServer;
       LatestReading = new SensorBean();
   }

   @Override
   public void run() {

         BufferedReader bufferedReader;
         PrintWriter printWriter;
       try {
           // get user input and output them
         bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         printWriter = new PrintWriter(socket.getOutputStream(), true);

         while (true) {
               //get the sensor name
                     printWriter.println("GETNAME");
               givenSensor = bufferedReader.readLine();
               if (givenSensor == null) {
                     System.out.println("Please enter a valid sensor
login name!");
                  return;
               }
               //Check whether entered name already exists or not.i.e :
threadsafe
               synchronized (sensorTypes) {
                   if (!sensorTypes.contains(givenSensor)) {
                     sensorTypes.add(givenSensor);
                       break;}}}
        // Assign passwords to sensors
         while (true){
               printWriter.println("GETPASSWORD");
               password = bufferedReader.readLine();
               if (password == null) {
                     System.out.println("Please enter a valid
password!");
                  return;
               }
               // credentials are validated
               if(authenticateSensorUser(givenSensor, password)){
                   break;
               }
            }

         // Assign floor id
         while (true) {
```

```
                printWriter.println("GETFLOOR");
                floor = Integer.parseInt(bufferedReader.readLine());
                if (floor == 0) {
                      System.out.println("Please enter a valid Floor
no!");
                    return;
                }
                sensor.setFloor(floor);
                break;
             }

            //Notify successfull registration to sensors
            printWriter.println("REGISTEREDSENSOR")

            //Get Readings from the sensors
            while (true) {
                 input = bufferedReader.readLine();
                 if (input == null) {
                     return;
                 }
                 // Deserializing the records to a byte stream
                 try(ObjectInputStream objc = new ObjectInputStream
                         (new
ByteArrayInputStream(Base64.getDecoder().decode(input)))){
                   LatestReading = (SensorBean) objc.readObject();
                     sensor.setLatestReading(LatestReading);

                     // Validate critical conditions
                     validateAlertConditions(sensor);
                 }
            }
       }
         finally {
           // Removing existing sensor if failure occurs
            fireSensorServer.removeExistingSensor(sensor);
           // Disconnected sensors are removed
           sensorTypes.remove(givenSensor);
           try {
               //close the connection
                  socket.close();
            } catch (IOException ex) {
                  System.err.println(ex.getMessage());
            }

       }
   }

   public static boolean authenticateSensorUser(String username, String
password){
            String credentials;
            String[] data;
            String uname;
            String pword;
       // Get data from text file
       try (BufferedReader br = new BufferedReader(new
InputStreamReader
```

```
        // credentials are taken from text file
        (SensorController.class.getResourceAsStream("SensorLogins.txt")))
) {
            credentials = br.readLine();
            while ( credentials!= null) {
                data = credentials.split("-");
                uname = data[0];
                pword = data[1];
                if(uname.equals(username) && pword.equals(password)){
                        return true;
                }
            }

            return false;
        }
        return false;
    }
  //critical conditions are checked
   public void validateAlertConditions(Sensor sensor){
         double temp = sensor.getLatestReading().getTemperature();
         int smoke = sensor.getLatestReading().getSmokeLevel();
             //alert situations
         if( temp > 50 ||  smoke > 7){
             System.out.println("In a critical Situation");
             //Allow sensor to make a alert
             fireSensorServer.makeAlerts(sensor);
         }

   }}
```

FireMonitorGUI.java

```
/**This class contain the GUI components of Monitor

public class FireMonitorGUI extends JFrame {
    private JButton btnUpdate;
    private JButton btnGetReading;
    private JLabel lblMon;
    private JLabel lblSen;
    private JLabel cmbFloor;
    private JLabel lblImg;
    private JPanel readingPanel;
    private JScrollPane scrollPanel;
    private JComboBox<Integer> floorno;
    private JLabel monitorCount;
    private JLabel sensorCount;
    private int floor;
    private JLabel lblSensor;

    private static FireMonitorController fireMonitor;

    public FireMonitorGUI(FireMonitorController fireMonitor) {
        initComponents();
        this.fireMonitor = fireMonitor;

    }
    //setting up monitor count
    public void setMonitorCount(int monitorCount){
        this.monitorCount.setText(Integer.toString(monitorCount));
```

```
}
//setting up sensor count
public void setSensorCount(int sensorCount){
    sensorCount.setText(Integer.toString(sensorCount));
}
//update new monitor count in the gui
public void addNewMonitor(SensorMonitorGUI sensorMonitor){
    readingPanel.add(sensorMonitor);
    sensorMonitor.setVisible(true);
}
 //add floor no to combo box
public void addToFloor(int floor){
    floorno.addItem(floor);
}

//initialize gui components
private void initComponents() {
    lblMon = new JLabel();
    lblSensor = new JLabel();
    scrollPanel = new JScrollPane();
    readingPanel = new JPanel();
    cmbFloor = new JLabel();
    lblImg = new JLabel();
    btnUpdate = new JButton();
    lblSen = new JLabel();
    monitorCount = new JLabel();
    sensorCount = new JLabel();
    floorno = new JComboBox<>();
    btnGetReading = new JButton();

    //setting up GUI components of firemonitor
    setMaximumSize(new Dimension(1100, 600));
    setMinimumSize(new Dimension(1100, 600));
    setPreferredSize(new Dimension(1100, 600));
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setResizable(false);
    setSize(new Dimension(1100, 600));
    getContentPane().setLayout(null);
    setTitle("Fire Monitor System Control Panel");

    readingPanel.setLayout(new GridLayout(1, 11, 111, 151));
    readingPanel.setBackground(new Color(12, 110, 265));
    scrollPanel.setViewportView(readingPanel);

    getContentPane().add(scrollPanel);
    scrollPanel.setBounds(20, 110, 1020, 380);
    scrollPanel.setBackground(new Color(29, 49, 162));

    lblMon.setFont(new java.awt.Font("Arial", 0, 15));
    lblMon.setForeground(new Color(255, 255, 215));
    lblMon.setText("Monitor Count");
    getContentPane().add(lblMon);
    lblMon.setBounds(20, 30, 170, 21);

    lblSen.setFont(new java.awt.Font("Arial", 0, 15));
    lblSen.setForeground(new Color(255, 255, 215));
    lblSen.setText("Sensor Count");
```

```
        getContentPane().add(lblSen);
        lblSen.setBounds(250, 30, 190, 21);

        lblSensor.setFont(new java.awt.Font("Arial", 0, 15));
        lblSensor.setForeground(new Color(255, 255, 215));
        lblSensor.setText("Sensors");
        getContentPane().add(lblSensor);
        lblSensor.setBounds(840, 30, 190, 21);

        monitorCount.setFont(new java.awt.Font("Arial", 0, 21));
        monitorCount.setForeground(new Color(215, 255, 215));
        monitorCount.setText("0");
        getContentPane().add(monitorCount);
        monitorCount.setBounds(70, 30, 50, 75);

        btnUpdate.setText("Latest Reading");
        btnUpdate.setBackground(new Color(189, 28, 230));
        btnUpdate.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                btnUpdateActionPerformed(evt);
            }
        });

        sensorCount.setFont(new Font("Arial", 0, 30));
        sensorCount.setForeground(new Color(255, 255, 255));
        sensorCount.setText("4");
        getContentPane().add(sensorCount);
        sensorCount.setBounds(300, 55, 20, 21);
        getContentPane().add(btnUpdate);
        btnUpdate.setBounds(700, 55, 164, 29);

        btnGetReading.setBackground(new Color(189, 28, 230));
        btnGetReading.setText("Get New Monitors");
        btnGetReading.addActionListener(new
java.awt.event.ActionListener() {
            // action of update button is defined
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                btnGetReadingActionPerformed(evt);
            }
        });
        getContentPane().add(btnGetReading);
        btnGetReading.setBounds(880, 55, 164, 29);

        getContentPane().add(floorno);
        floorno.setBounds(450, 56, 181, 27);

        cmbFloor.setFont(new Font("Arial", 0, 18));
        cmbFloor.setForeground(new Color(255, 255, 255));
        cmbFloor.setText("Select Floor ID");
        getContentPane().add(cmbFloor);
        cmbFloor.setBounds(480, 30, 130, 21);
```

```
        lblImg.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/com/sliit/images/fire.jp
g")));
        getContentPane().add(lblImg);
        lblImg.setBounds(0, 0, 1100, 590);

        pack();
    }
    //Readings are updated
    private void btnUpdateActionPerformed(java.awt.event.ActionEvent
evt) {
        fireMonitor.updateExistingReadings();
    }

    public static void main(String args[]) throws
ClassNotFoundException, IllegalAccessException,
UnsupportedLookAndFeelException {

        try {
            for (LookAndFeelInfo data:
UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(data.getName())) {
                    = UIManager.setLookAndFeel(data.getClassName());
                    break;
                }
            }
        }

        EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FireMonitorGUI(fireMonitor).setVisible(true);
            }
        });
    }
}
```

UserLoginMonitorGUI.java

```
/**This class contains the login GUI components

public class UserLoginMonitorGUI extends JFrame {

          private JButton btnLogin;
          private JLabel lblImage;
          private JPasswordField password;
          private JTextField username;
          private MonitorApplication monitorApp;

          public UserLoginMonitorGUI() {
          initComponents();

        //initialize the application
        monitorApp = new MonitorApplication();
    }


    @SuppressWarnings("unchecked")
    private void initComponents() {
```

```
        username = new JTextField();
        btnLogin = new JButton();
        password = new JPasswordField();
        lblImage = new JLabel();

        //setting up dimensions of logging screen
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setForeground(new Color(204, 0, 204));
        setLocationByPlatform(true);
        setPreferredSize(new Dimension(700, 400));
        setResizable(false);
        setName("Login");
        setTitle("Login To Fire Monitor System");
        getContentPane().setLayout(null);
        getContentPane().add(username);
        username.setBounds(460, 130, 190, 26);

        btnLogin.setText("Login");
        btnLogin.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(ActionEvent evt) {

try {
      btnLoginActionPerformed(evt);
      } catch (HeadlessException | IOException e) {
                  System.out.println(e.getMessage());
                              }

            }
        });
        getContentPane().add(btnLogin);
        btnLogin.setBackground(new Color(59, 89, 182));
        btnLogin.setBounds(500, 200, 110, 40);
        getContentPane().add(password);
        password.setBounds(460, 160, 190, 26);
        //background image
        lblImage.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/com/sliit/images/fire1.j
pg")));
        getContentPane().add(lblImage);
        lblImage.setBounds(0, 0, 700, 400);

        pack();
    }
    //action of logging button
    private void btnLoginActionPerformed(java.awt.event.ActionEvent
evt) throws HeadlessException, IOException {

if(LoginController.authenticateUser(username.getText(),password.getText
())){monitorApp.intializeMonitor();
            this.dispose();
        }
        else{
            JOptionPane.showMessageDialog(
                                    null,
```

```
                                        "Authentication Failed! Please
check your username and password",
                                        "Login Failed",
                                        JOptionPane.ERROR_MESSAGE);
            }
        }

        public static void main(String args[]) {
            try {
                for (UIManager.LookAndFeelInfo data :
UIManager.getInstalledLookAndFeels()) {
                    if ("Nimbus".equals(data.getName())) {
UIManager.setLookAndFeel(data.getClassName());
                        break;
                    }
                }
            } catch (ClassNotFoundException ex) {
            } catch (InstantiationException ex) {
            } catch (IllegalAccessException ex) {
            } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            }

            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    new UserLoginMonitorGUI().setVisible(true);
                }
            });
        }

}
```

BatteryLevel.java

```
/**This class contains reading of batterylevel
public class BatteryLevel {

    private double batteryLevel;

    public BatteryLevel() {
      //define initial value
      batteryLevel = 1;
    }


    public double generateReading(){
      Random random = new Random();
      // generate a random integer from 0 to 100
      int value = random.nextInt(100);
      if (value < 0) {
            batteryLevel += 10;
            return batteryLevel;
      }
      return (batteryLevel+value);
    }

}
```

CO2Level.java

```
/**This class contains reading of co2level

public class CO2Level {

    private double co2Level;

    public CO2Level() {
      //define initial value
      co2Level = 1;
    }


    public double generateReading(){
      Random random = new Random();
      // generate a random integer from 0 to 1000
      int value = random.nextInt(1000);
        if (value < 0) {
            co2Level += 10;
            return co2Level;
        }
        return co2Level+value;
    }

}
```

SmokeLevel.java

```
/**This class contains reading of smokelevel

public class SmokeLevel {
    private int smokeLevel;

    public SmokeLevel() {
      //define initial value
      smokeLevel = 1;
    }

    public double generateReading(){
      Random random = new Random();
      // generate a random integer from 0 to 10
      int value = random.nextInt(10);
        if (value < 0) {
            smokeLevel += 1;
            return smokeLevel;
        }

        return smokeLevel+value;
    }
}
```

Temperature.java

```
/**This class contains reading of temperature

public class Temperature{

    private double temperature;
```

```
    public Temperature() {
      //define initial value
      temperature = 4;
    }


    public double generateReading(){

            Random random = new Random();
            // generate a random integer from 0 to 90
            int value = random.nextInt(90);
            if (value < 0) {
                  temperature += 1.5;
                  return temperature;
            }

        return temperature+value;
    }

}
```

SensorListener.java

```
/** Interface which notifies all the monitors with callback methods

public interface SensorListener extends Remote{
    //get the updated readings
    public void ModifyReading(List<Sensor> sensors, int monitorCount,
int sensorCount) throws RemoteException;
    //notify on alert situations
    public void notifyAlertSituation(Sensor sensor) throws
RemoteException;
  //notify on failure situations
    public void makeAlertsOnFailures(Sensor sensor) throws
RemoteException;
}
```

Sensor.java

```
/** This act as bean which contain the reading data

public class Sensor implements Serializable{
    private int floor;
    private SensorBean latestReading;

    public Sensor() {}
    //getter to get the floor no
    public int getFloor() {
        return floor;
    }
  //setter to set the floor no
    public void setFloor(int floor) {
        this.floor = floor;
    }
    //getter to get the latest reading
    public SensorBean getLatestReading() {
        return latestReading;
```

```
    }

    //setter to set the latest reading
    public void setLatestReading(SensorBean latestReading) {
        this.latestReading = latestReading;
    }

    //display latest reading with floor number
    @Override
    public String toString() {
        return "Reading of the Sensor\n"
       + "Floor ID is: " + floor + "\nLatest Reading is: " +
latestReading;
   }
}
```

FireSensor.java

```
/**This class use to add new sensors to the monitor after authentication

public class FireSensor  {

    private static FireSensor fireSensor;
    private Temperature temperature;
    private SmokeLevel smokeLevel;
    private CO2Level co2Level;
    private BatteryLevel batteryLevel;

    private SensorBean reading;
    private BufferedReader bufferedReader;
    private PrintWriter printWriter;
    private String server;
    private Socket socket;
    private String input;
    private String username;
    private String password;
    private Scanner userInput;
    private int floor;
    private long breakReading;

    public FireSensor() {
        userInput = new Scanner(System.in);
        temperature = new Temperature();
        smokeLevel = new SmokeLevel();
        batteryLevel = new BatteryLevel();
        co2Level = new CO2Level();
        reading = new SensorBean();

    }


    public void run() throws IOException, InterruptedException {

        // Socket Connection is created
        server = "127.0.0.1";
        socket = new Socket(server, 9001);
```

```
        //get user input and output
        bufferedReader = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
        printWriter = new PrintWriter(socket.getOutputStream(), true);

        // Sensor Authentication data is processed
        while (true) {
            input = bufferedReader.readLine();
            if (input.startsWith("GETNAME") && input != null ) {
                  printWriter.println(getUsername());
            } else if (input.startsWith("GETPASSWORD") && input !=
null){
                  printWriter.println(getPassword());
            } else if (input.startsWith("GETFLOOR") && input != null){
                  printWriter.println(getFloor());
            } else if (input.startsWith("REGISTEREDSENSOR")&& input !=
null){
                  generateSensorReading();
            }
        }
    }
   //get the sensor username
    private String getUsername() {
      try {
            System.out.print("Please Enter your sensor username : ");
            username = userInput.nextLine();
            return username;
            } catch (Exception e) {
                  return "Incorrect Username ";
            }
     }
    //get the sensor password
    private String getPassword(){
      try {
        System.out.print("Please Enter your sensor password: ");
        password = userInput.nextLine();
        return password;
      } catch (Exception e) {
                  return "Incorrect Username ";
            }

    }
    //get the floor id
    private int getFloor() {
        System.out.print("Please Enter Floor ID: ");
        floor = Integer.parseInt(userInput.nextLine());

        //validation to check valid input
        if(floor<1 || floor >50) {
            System.out.println("Please enter a valid floor number
between 1 and 50");
             System.out.print("Please Enter Floor ID: ");
             floor = Integer.parseInt(userInput.nextLine());
        }
        return floor;
    }
```

```
    private void generateSensorReading() throws IOException,
InterruptedException{
        for (; ;) {
            //setting up readings
            reading.setTemperature(temperature.generateReading());
            reading.setSmokeLevel((int) smokeLevel.generateReading());

      reading.setBatteryLevel(batteryLevel.generateReading());
      reading.setCO2Level(co2Level.generateReading());
      System.out.println(reading);


                    } catch (IOException ex) {
                        System.err.println("Serialization failed
"+ex.getMessage());
                      }
                    //to avoid continuous display
                     breakReading = 500000;
                     Thread.sleep(breakReading);
            }
    }
    //Initiate the fire sensor
    public static void main(String[] args) throws Exception {
        fireSensor = new FireSensor();
        fireSensor.run();}}
```

SensorMonitorGUI.java

```
/**This class contains GUI components of the monitor screen

public class SensorMonitorGUI extends JInternalFrame {
          private JTextField batterylevel;
          private JTextField smoke;
          private JLabel lblcentri;
          private JLabel lbltem;
          private JLabel lblsm;
          private JLabel lblco2;
          private JLabel lblbat;
          private JLabel lblsmoke;
          private JLabel lblperc;
          private JLabel lblppm;
          private JLabel floor;
          private JTextField Co2level;
          private JTextField temperature;

    public SensorMonitorGUI(int floor) {
        initComponents();
        floor.setText(String.valueOf(floor));
    }
    //Setting up values to respective labels
    public void setTemperature(Double temp){
        temperature.setText(Double.toString(temp));
    }

    public void setSmokeLevel(int smokeLevel){
        smoke.setText(Integer.toString(smokeLevel));
    }
```

```
    public void setCO2Level(double co2Level){
        batterylevel.setText(Double.toString(co2Level));
    }

    public void setBatteryLevel(double batteryLevel){
        Co2level.setText(Double.toString(batteryLevel));
    }

    //setting up GUI components
    @SuppressWarnings("unchecked")
    private void initComponents() {

        smoke = new JTextField();
        lbltem = new JLabel();
        lblsm = new JLabel();
        temperature = new JTextField();
        floor = new JLabel();
        batterylevel = new JTextField();
        lblco2 = new JLabel();
        lblbat = new JLabel();
        Co2level = new JTextField();
        lblcentri = new JLabel();
        lblsmoke = new JLabel();
        lblperc = new JLabel();
        lblppm = new JLabel();

        setMaximumSize(new java.awt.Dimension(300, 280));
        setSize(new java.awt.Dimension(280, 220));
        setBackground(new Color(224, 20, 244));
        smoke.setEditable(false);
        smoke.setName("smoke");
        smoke.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                smokeActionPerformed(evt);
            }
        });

        lbltem.setText("Temperature");

        lblsm.setText("Smoke Level");

        temperature.setEditable(false);
        temperature.setName("temperature");
        temperature.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                temperatureActionPerformed(evt);
            }
        });

    private void smokeActionPerformed(java.awt.event.ActionEvent evt) {
    }

    private void temperatureActionPerformed(java.awt.event.ActionEvent
evt) {
```

```
    }

    private void batterylevelActionPerformed(java.awt.event.ActionEvent
evt) {
    }

    private void Co2levelActionPerformed(java.awt.event.ActionEvent
evt) {    }

        //default value of floor set to 1
        floor.setText("1");

        batterylevel.setEditable(false);
        batterylevel.setName("batterylevel");
        batterylevel.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                batterylevelActionPerformed(evt);
            }
        });


        //labels
        lblco2.setText("CO2 Level");
        lblbat.setText("Battery Level");
        Co2level.setEditable(false);
        Co2level.setName("Co2level");
        Co2level.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt)
{
                Co2levelActionPerformed(evt);
            }
        });
        //setting up units in the labels
        lblcentri.setText("°C");
        lblsmoke.setText("");
        lblperc.setText("%");
        lblppm.setText("ppm");

        //setting up label styles
        lblcentri.setFont(new Font("Courier New", Font.ITALIC, 14));
        lblcentri.setForeground(Color.BLUE);
        lblsmoke.setFont(new Font("Courier New", Font.ITALIC, 14));
        lblsmoke.setForeground(Color.BLUE);
        lblperc.setFont(new Font("Courier New", Font.ITALIC, 14));
        lblperc.setForeground(Color.BLUE);
        lblppm.setFont(new Font("Courier New", Font.ITALIC, 14));
        lblppm.setForeground(Color.BLUE);

        lbltem.setFont(new Font("Arial", Font.BOLD, 15));
        lbltem.setForeground(Color.MAGENTA);
        lblsm.setFont(new Font("Arial", Font.BOLD, 15));
        lblsm.setForeground(Color.MAGENTA);
        lblbat.setFont(new Font("Arial", Font.BOLD, 15));
        lblbat.setForeground(Color.MAGENTA);
```

```
        lblco2.setFont(new Font("Arial", Font.BOLD, 15));
        lblco2.setForeground(Color.MAGENTA);

        lblfloorNo.setFont(new Font("Arial", Font.BOLD, 19));
        lblfloorNo.setForeground(Color.PINK);
    }
```