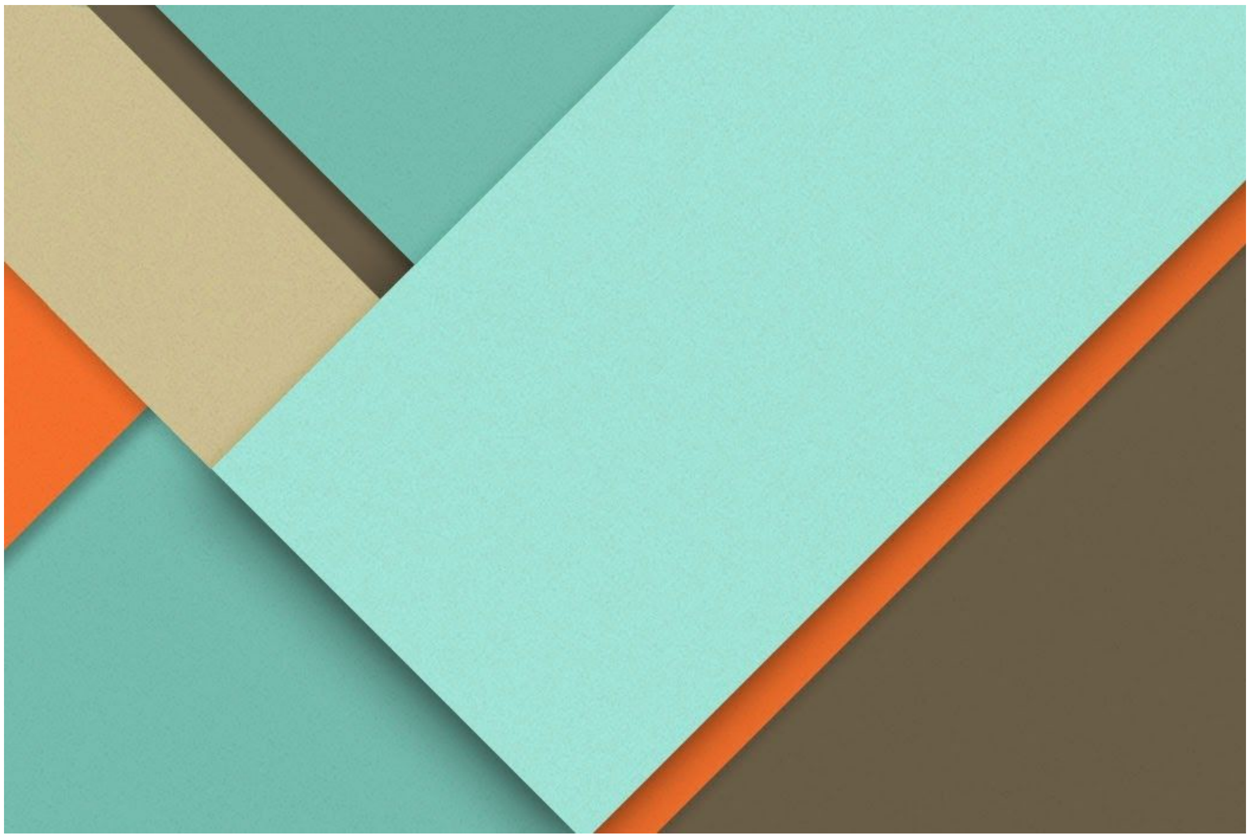


Fire Alarm System

20.11.2020

GROUP-12

Sankalp Rajendran	IIT2019173
Ayush Baranwal	IIT2019169
Sahil Sharma	IIT2019179
Aadharsh Roshan	IIT2019161

INDEX

Overview

In this project we have created a Fire Alarm System which would simulate the real time monitoring of Smoke, Heat and Carbon Monoxide sensors in numerous locations of the CC3 building by providing the functionality of registering sensors and configuring the system. It will also generate a log file for each session at an user defined interval.

Description

There are 6 floors in CC3 including the ground floor. We have assumed 11 locations at each floor. The user registers each sensor with unique ID at different locations and configures values such as threshold value, log interval, volume of alarm and email ID to send alerts to.
Random values are generated for each sensor around the configured thresholds and values are displayed in real time in the Monitoring screen along with alerts when current values breach threshold or a manual alarm is randomly raised.

Program Ability (Objectives)

- Register a new sensor from the user provided Sensor ID, Sensor Type, Floor No., Install Location.
- Configure sensor from the user of all three types: CO, Heat, Smoke provided Threshold Value, Duration of Alarm, Volume of Alarm, Log Interval, Email ID.
- Monitor the values at all the 6 floors each with locations Rooms 1 to 6, Lab 1, Lab 2 , Stairs 1, Stairs 2, and Hall.
- Show values in red colour if the threshold of its type is breached.
- Display the location and type of the triggered sensor with a timestamp and alert message when the threshold is breached.
- Send an email alert within 30 seconds of the breach of threshold and check for breaches every minute.
- Generate log when a new sensor is registered, a sensor is configured, a mail is sent and values of all sensors at various locations at the log interval.

Technologies used:

1. Java :

   Was used to demonstrate Object oriented programming principles and methodology.

2. Java Swing:.

   A potent and easy to use library to design GUI with Java.

3. Java Mail API:

   Used to send email alerts from the application.

4. Maven:

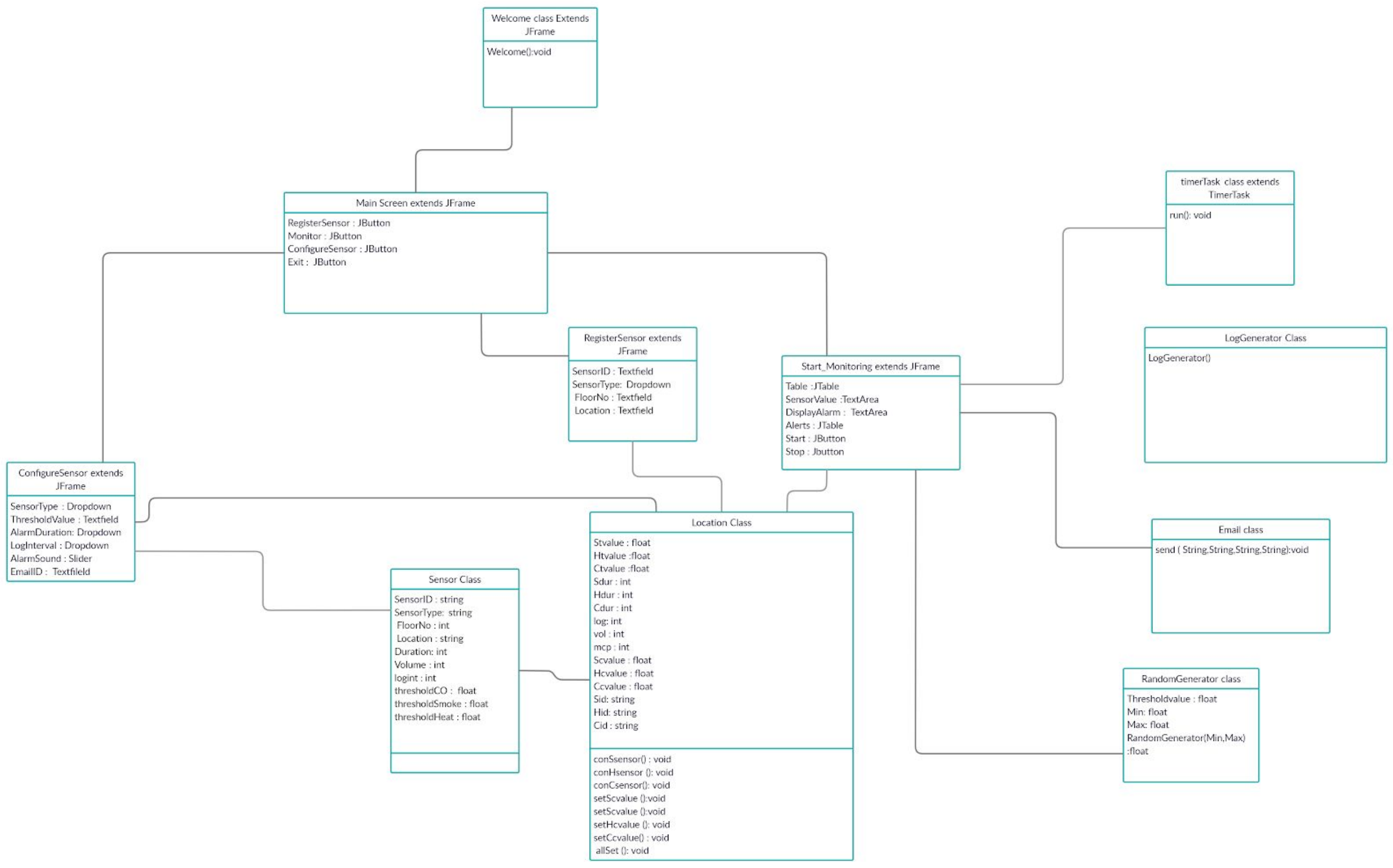
   Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Used to build the JAR file.

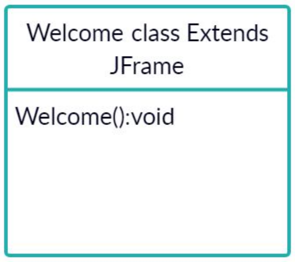
5. Git version control

   Git is a distributed version-control system for tracking changes in source code during software development. It was used for coordinated development of the project.

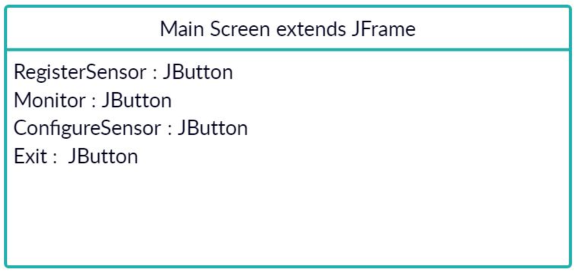
UML DIAGRAMS

- ## CLASS DIAGRAM

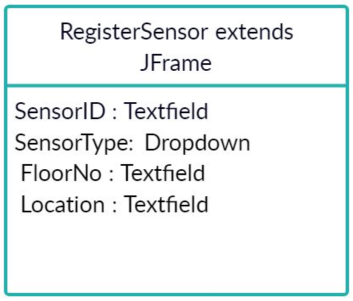




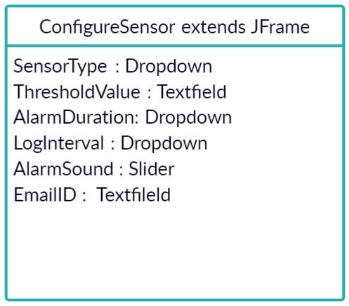
Welcome() : opens the welcome page . Also creates a mainscreen



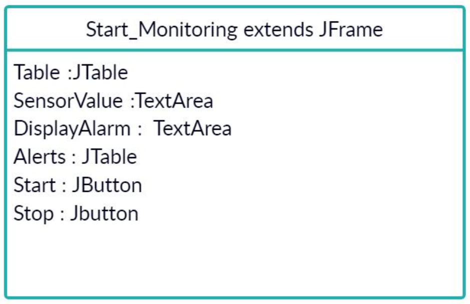
RegisterSensor - Register a new sensor
Monitor - Start monitoring of registered sensors
Configure sensor - Configure the registered sensors
Exit - Exit the system



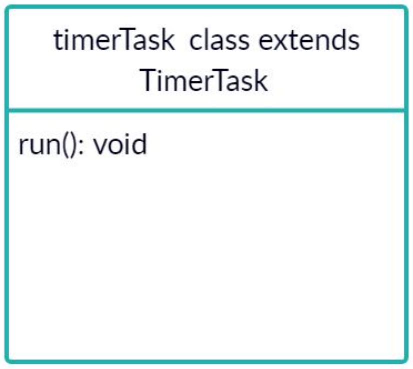
It has 4 attributes : SensorID , SensorType , FloorNo and Location



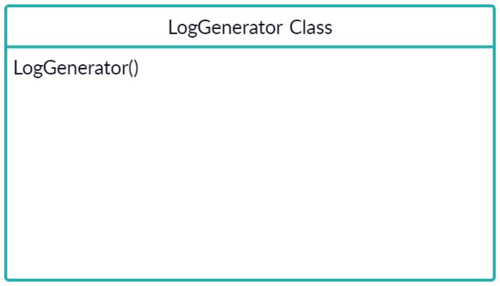
It has 6 attributes : SensorType , ThresholdValue , AlarmDuration , logInterval , AlarmSound and Email ID



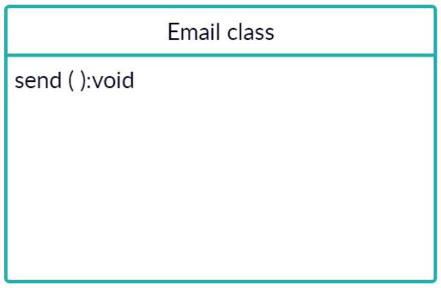
It has 2 attributes : SensorValue and DisplayAlarm
Start() - Starts the monitoring
Stop() - Stops the monitoring



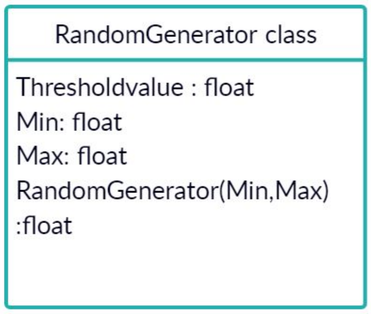
run() - Runs the given task according to the input time interval



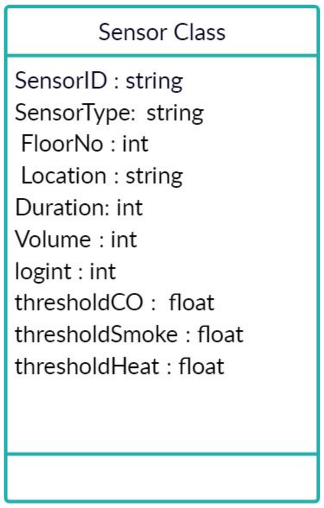
LogGenerator() - Generates Log report of the current values of sensors



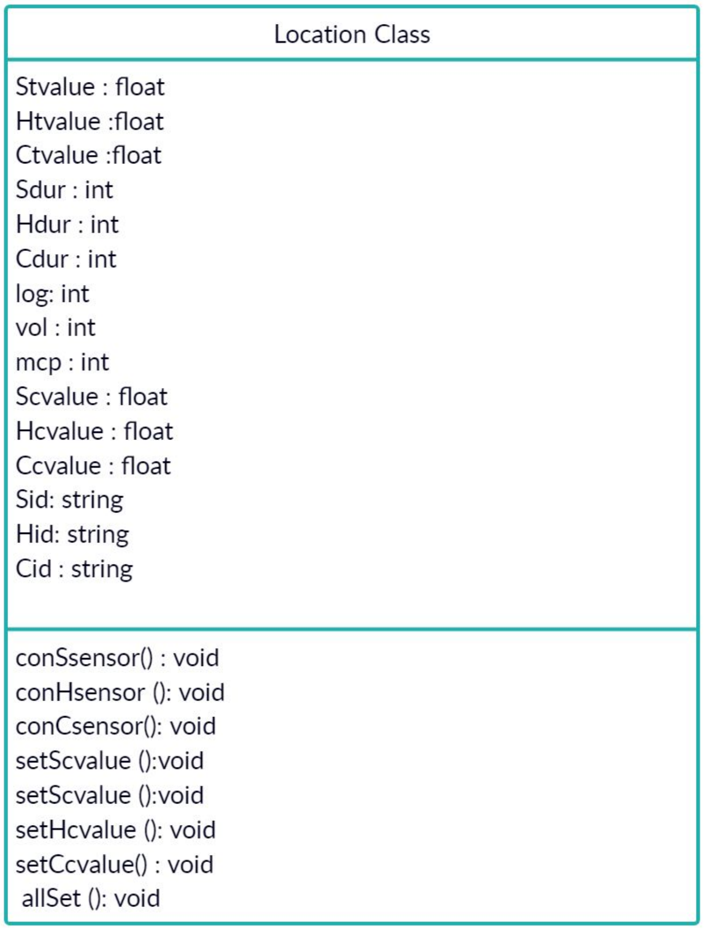
send() - Send email to the registered address



The attributes are Threshold , Min and Max
RandomGenerator() - Generates random values between Min and Max



It has 10 attributes : SensorID , Sensor Type , FloorNo , Location , Duration , Volume , Logint , thresholdCO , thresholdSmoke , thresholdHeat .



Location Class

Stvalue : float
Htvalue :float
Ctvalue :float
Sdur : int
Hdur : int
Cdur : int
log: int
vol : int
mcp : int
Scvalue : float
Hcvalue : float
Ccvalue : float
Sid: string
Hid: string
Cid : string

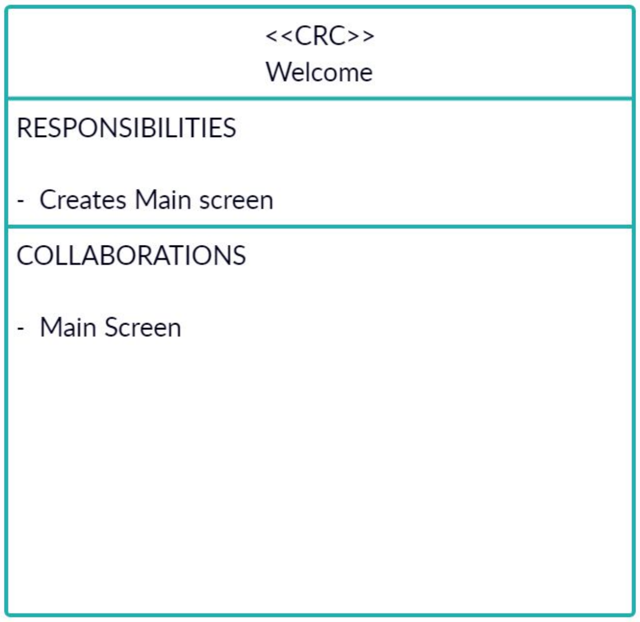
conSsensor() : void
conHsensor (): void
conCsensor(): void
setScvalue ():void
setScvalue ():void
setHcvalue (): void
setCcvalue() : void
allSet (): void

It has 15 attributes : Stvalue , Htvalue , Ctvalue , Sdur , Hdur , Cdur , log , vol , mcp , Scvalue , Hcvalue , Ccvalue , Sid , Hid , Cid

conSsensor - Set the values of Stvalue , Sdur , log , and vol
conCsensor - Set the values of Ctvalue , Cdur , log and vol
conHsensor - Set the values of Htvalue , Hdur , log and vol
setScvalue - Set the value of Scvalue
setHcvalue - set the value of Hcvalue
setCcvalue - set the value of Ccvalue
allSet() - Store the values of sensors of each floor in their respective maps

● CRC DIAGRAM

- Welcome Class



<<CRC>>
Welcome

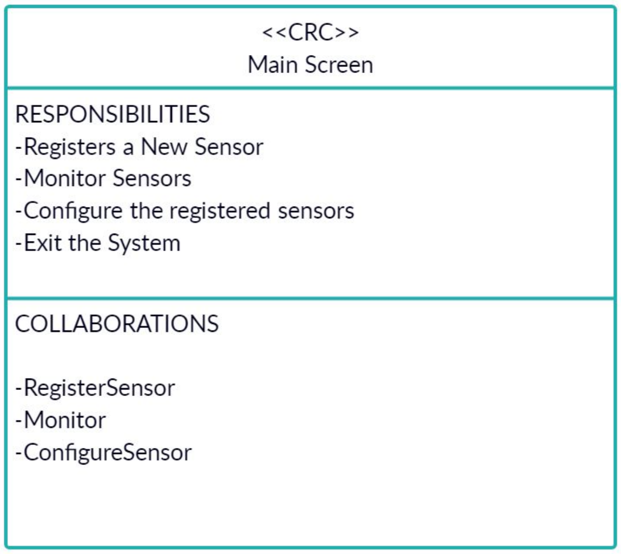
RESPONSIBILITIES

- Creates Main screen

COLLABORATIONS

- Main Screen

- MainScreen Class



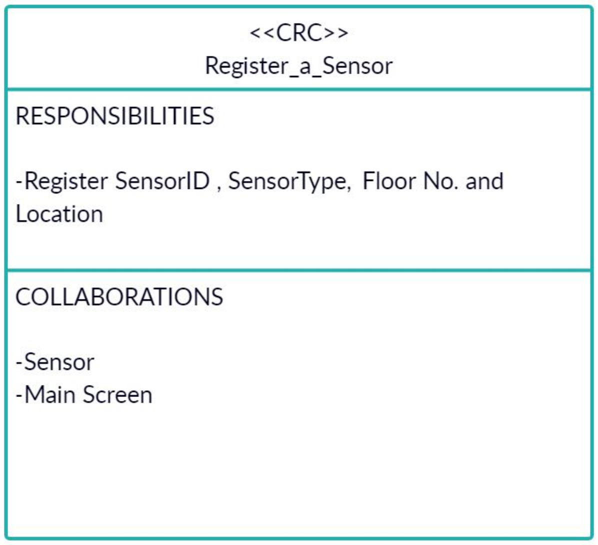
<<CRC>>
Main Screen

RESPONSIBILITIES
-Registers a New Sensor
-Monitor Sensors
-Configure the registered sensors
-Exit the System

COLLABORATIONS

-RegisterSensor
-Monitor
-ConfigureSensor

- Register_a_Sensor class



<<CRC>>
Register_a_Sensor

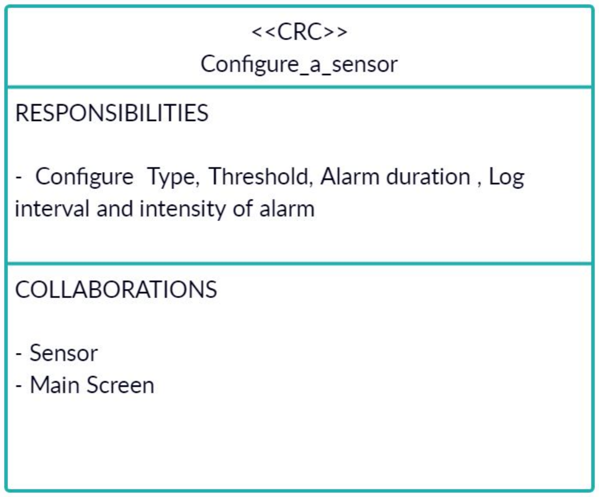
RESPONSIBILITIES

-Register SensorID , SensorType, Floor No. and Location

COLLABORATIONS

-Sensor
-Main Screen

- Configure_a_Sensor class



<<CRC>>
Configure_a_sensor

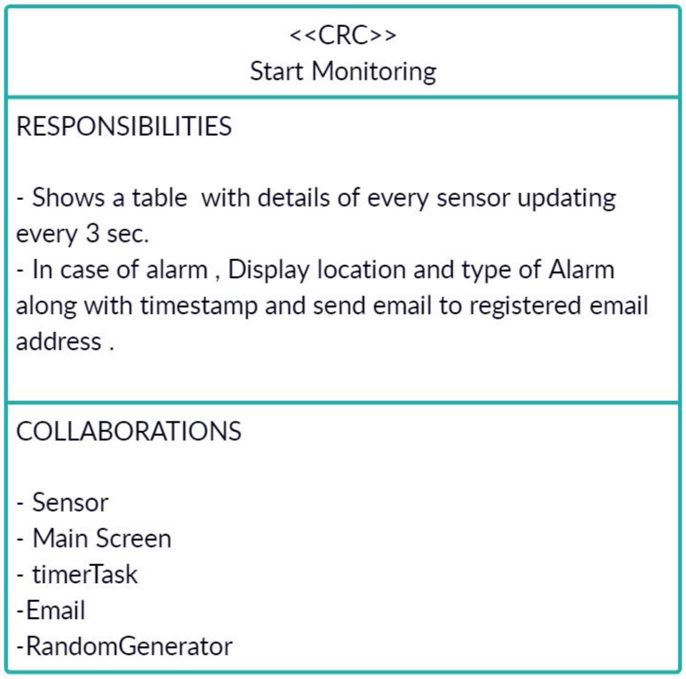
RESPONSIBILITIES

- Configure Type, Threshold, Alarm duration , Log interval and intensity of alarm

COLLABORATIONS

- Sensor
- Main Screen

- Start_Monitoring class



<<CRC>>
Start Monitoring

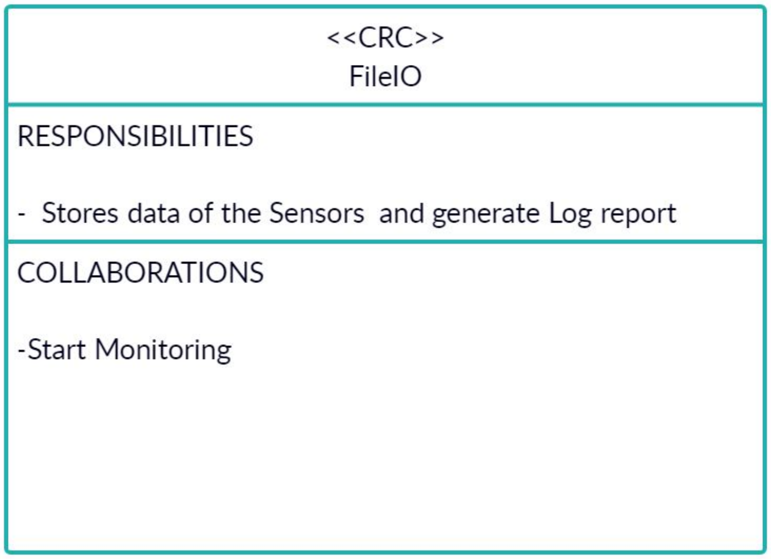
RESPONSIBILITIES

- Shows a table with details of every sensor updating every 3 sec.
- In case of alarm , Display location and type of Alarm along with timestamp and send email to registered email address .

COLLABORATIONS

- Sensor
- Main Screen
- timerTask
-Email
-RandomGenerator

- FileIO class



<<CRC>>
FileIO

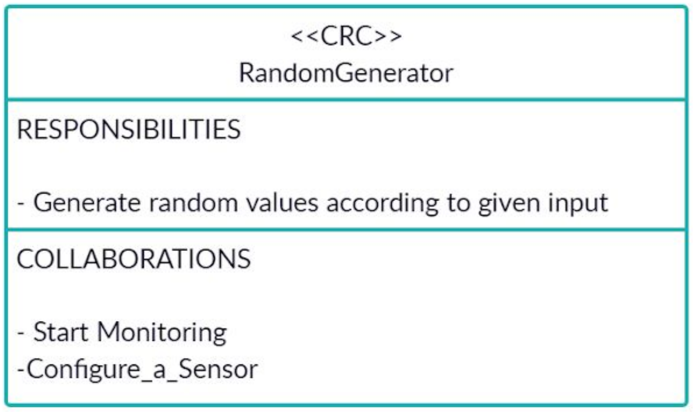
RESPONSIBILITIES

- Stores data of the Sensors and generate Log report

COLLABORATIONS

-Start Monitoring

- RandomGenerator class


<<CRC>>
RandomGenerator

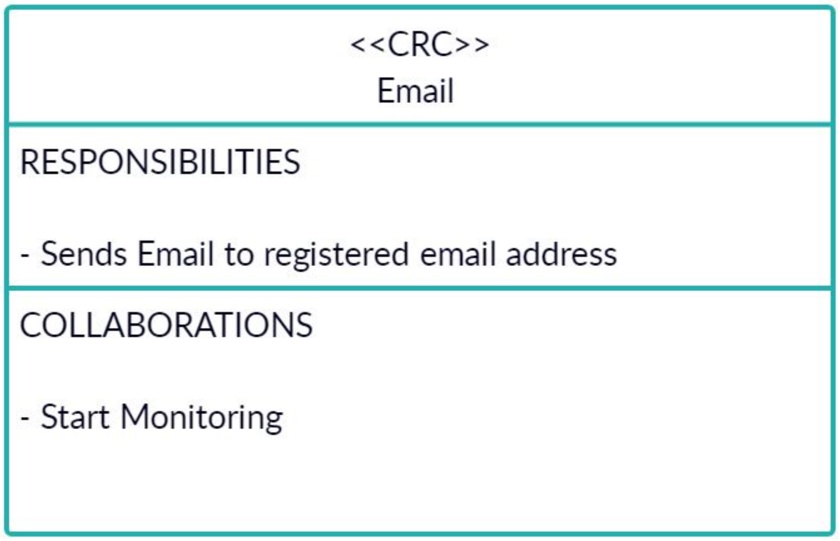
RESPONSIBILITIES

- Generate random values according to given input

COLLABORATIONS

- Start Monitoring
-Configure_a_Sensor

- Email class


<<CRC>>
Email

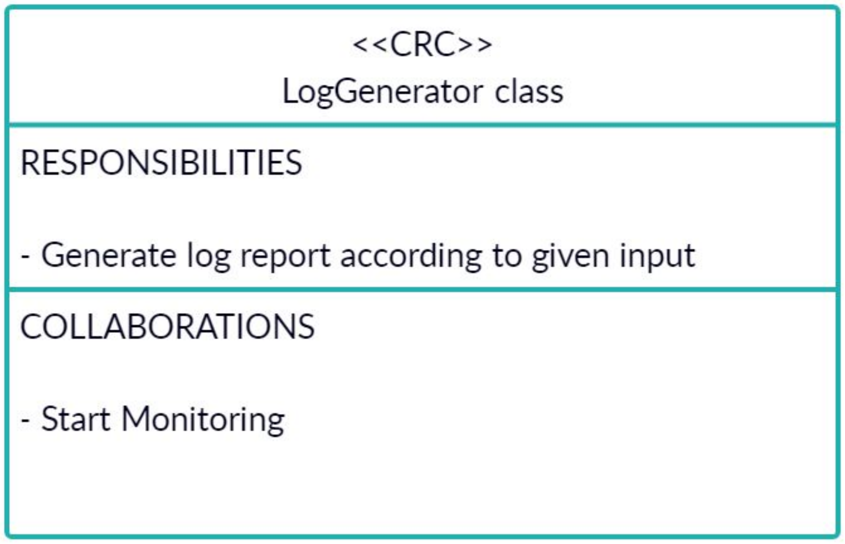
RESPONSIBILITIES

- Sends Email to registered email address

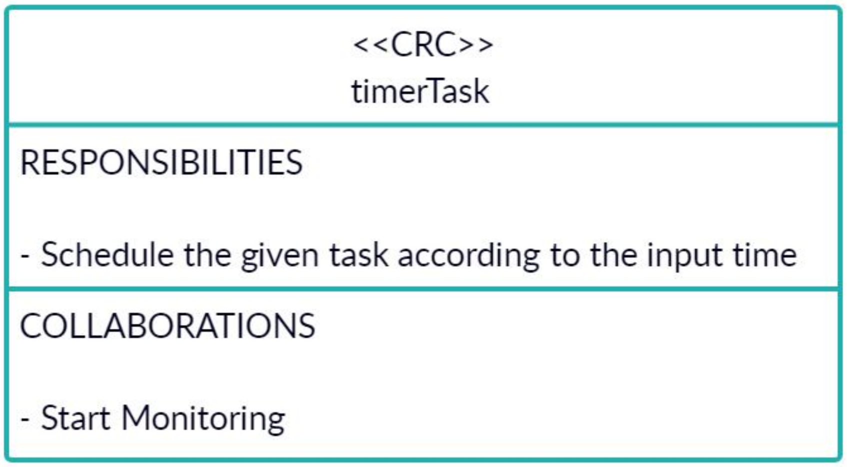
COLLABORATIONS

- Start Monitoring

- LogGenerator Class



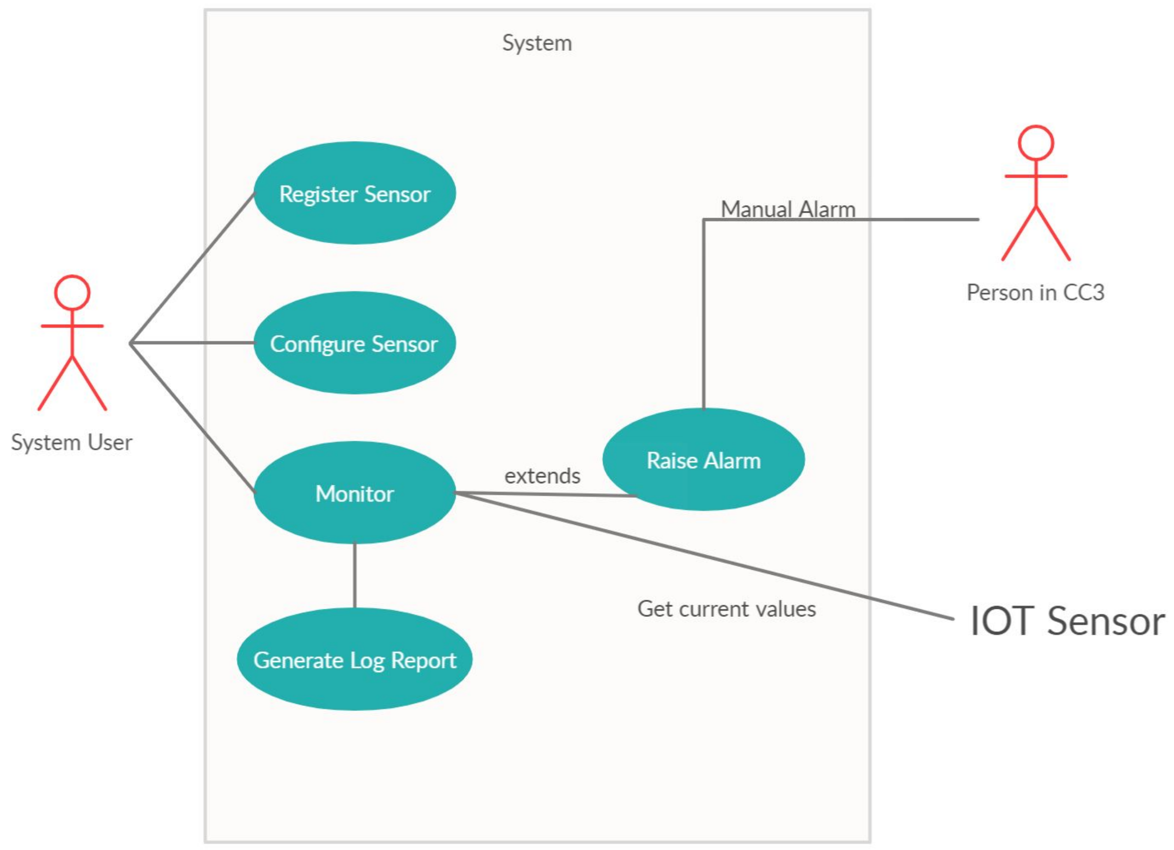
- timerTask class



• USE CASE DIAGRAMS

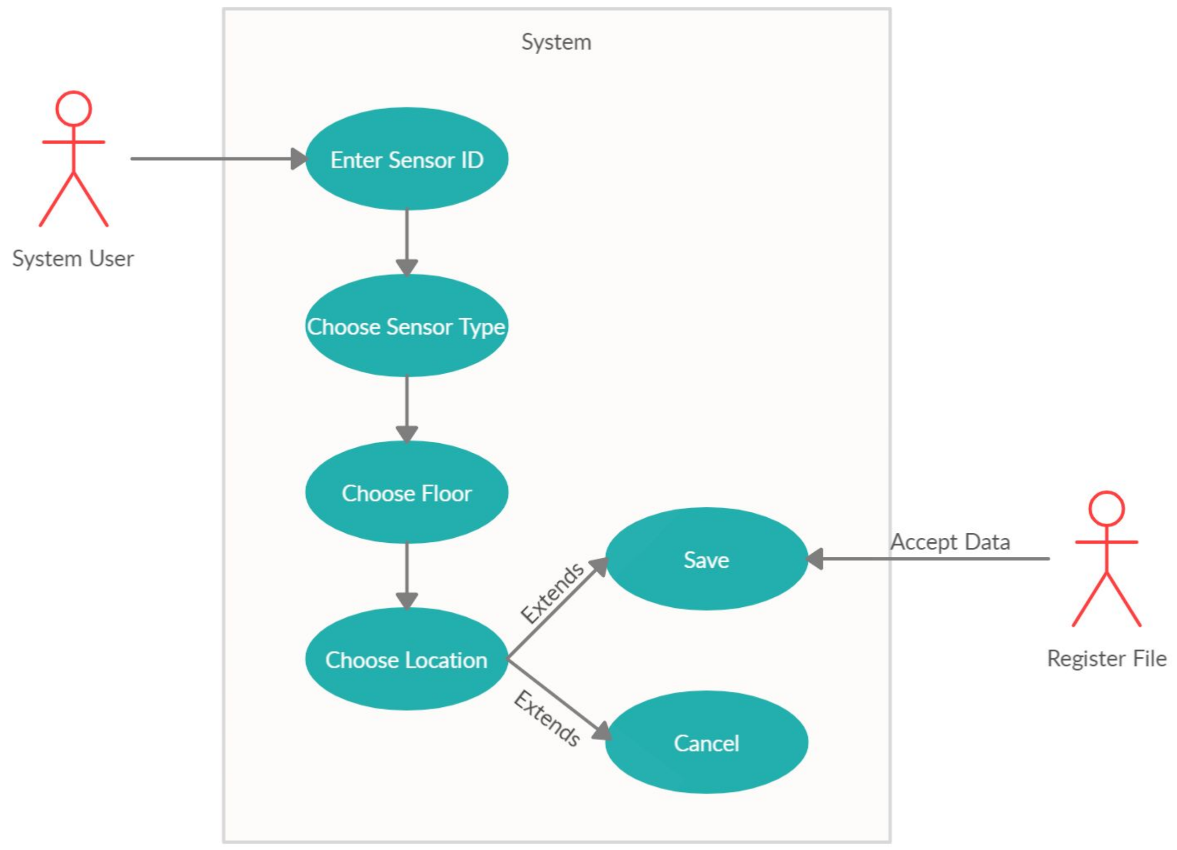
A use case diagram is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases.

Overall Use case Diagram-



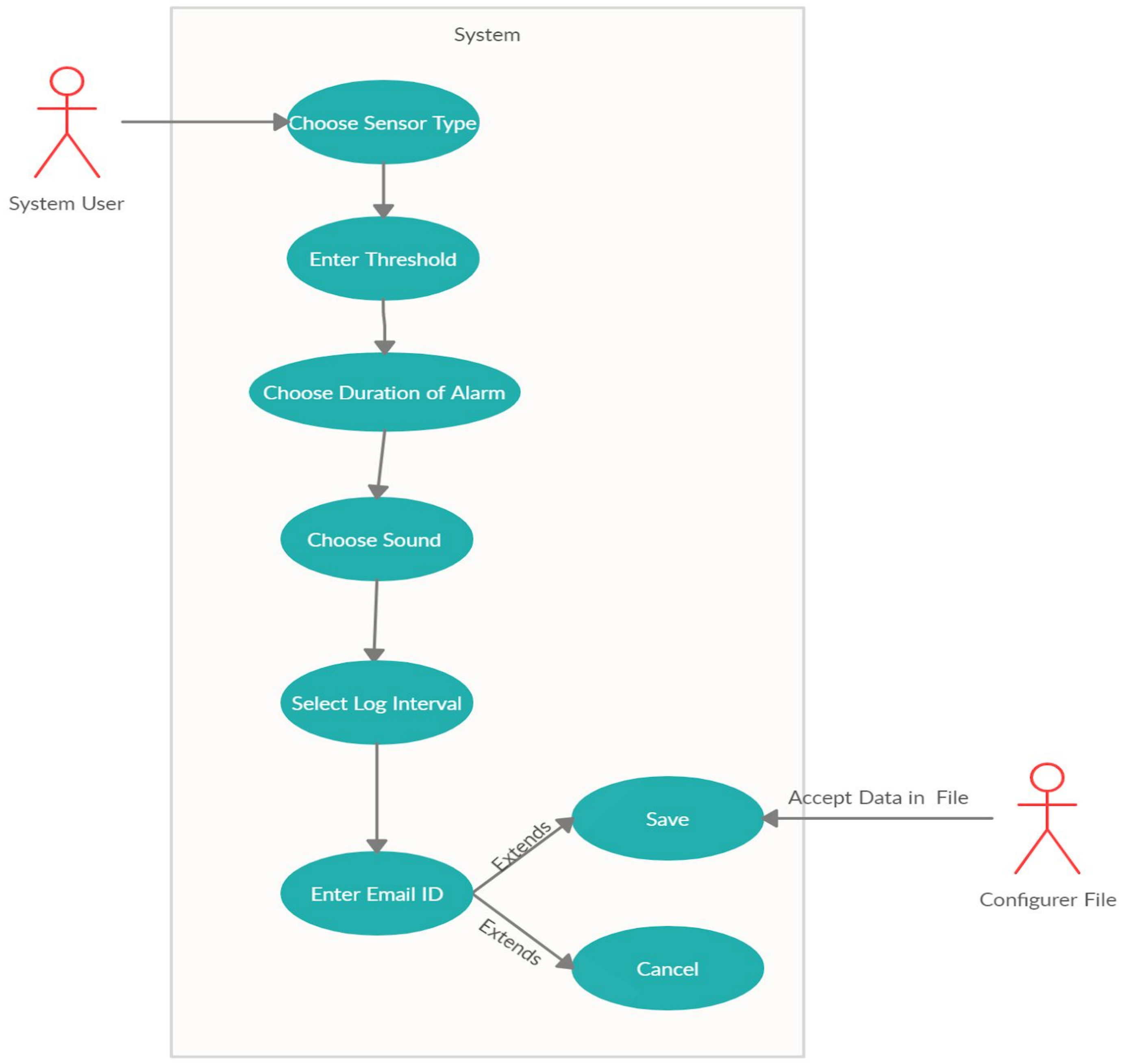
This is the use case diagram for using the system. It shows that the System user can register a sensor, configuring the types of sensor and monitoring the real time values of registered sensors. It shows that the system generates a log report in the background for assessment of current session.
A person in CC3 can also contribute to the use case as triggering the manual alarm also shows up as alerts in the system.
In an ideal case if IOT sensors are connected to the system then values are taken from them.

Register Use case Diagram-



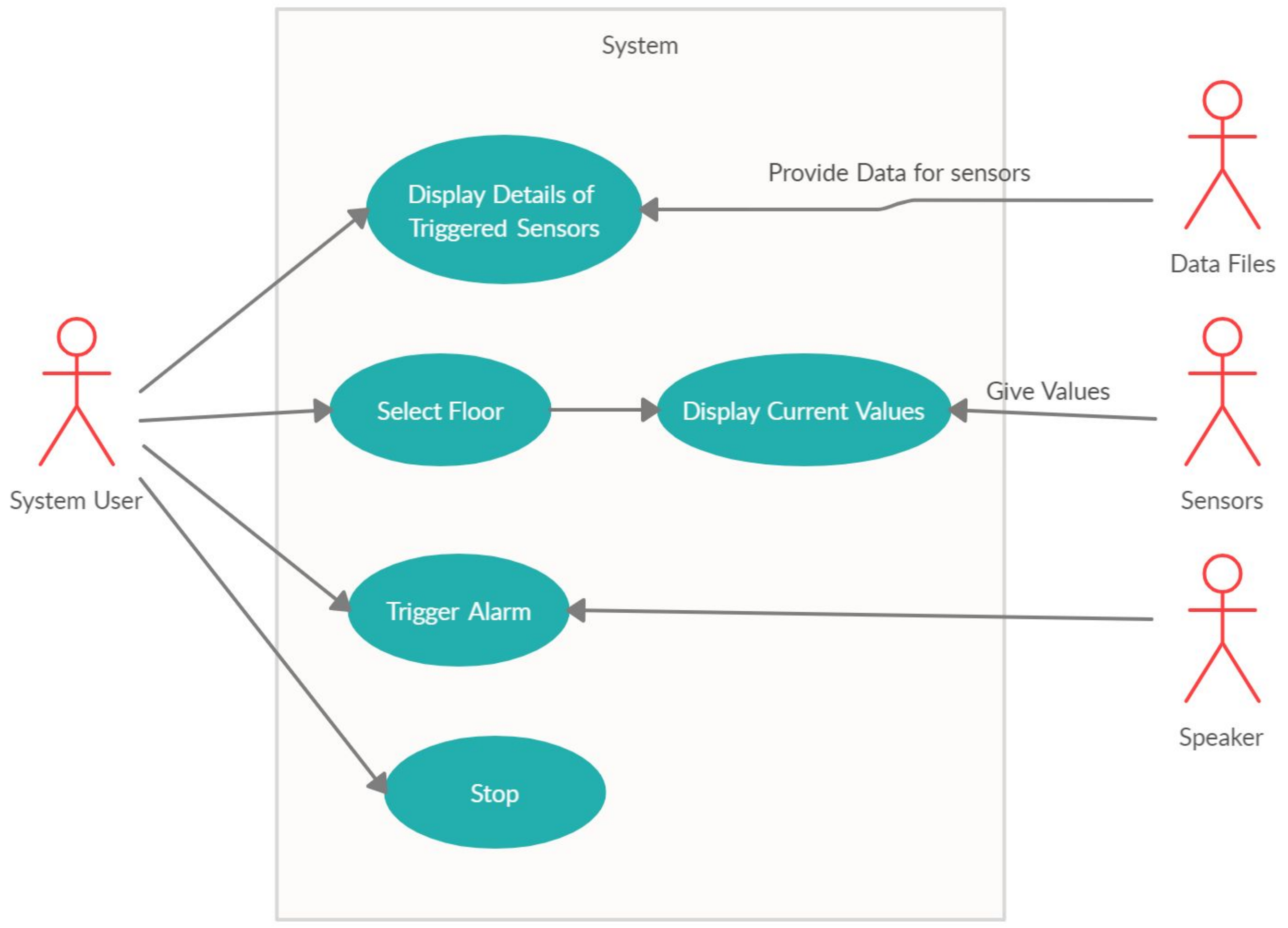
In this use case diagram, it shows that the user has to register each sensor by providing an ID, type of sensor, floor and location. Then it can be either saved or cancelled by back button. The data is accepted and stored in floor hashmaps.

Configure Use case Diagram-



In this use case diagram, the system user utilises the system by entering thresholds for different types of sensors and some overall settings like volume, log interval and email ID to receive alerts. The data can be saved or cancelled. The data is saved in floor hashmaps.

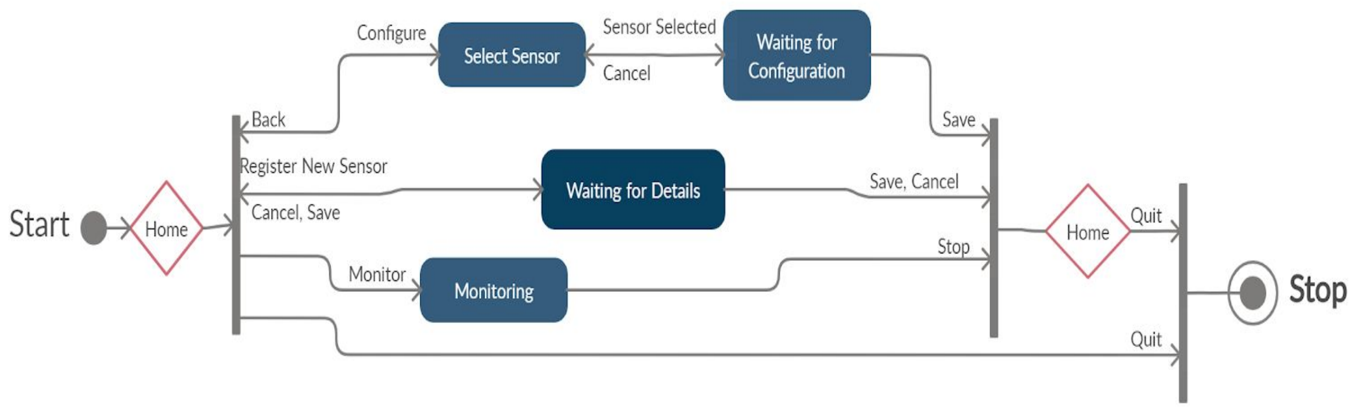
Monitor Use case Diagram-



In this use case diagram, the user can perform tasks such as monitoring sensor values at each location of all floors of CCR. Also alerts are displayed for breaches of threshold at any location. Stop button is for stopping these real time processes and going back to the dashboard.

• State Diagram

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions.

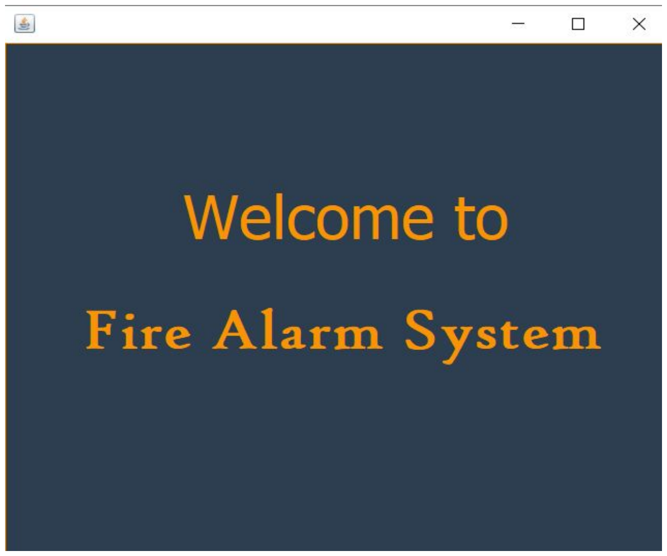


This diagram basically shows the states in which the program transitions at pressing of different buttons.

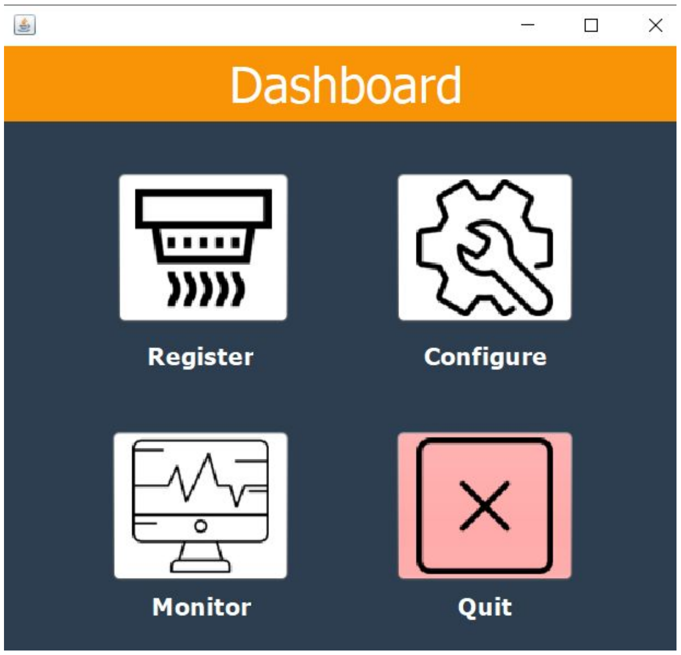
Frontend

Simulation

1. After executing the jar file a welcome screen is displayed.

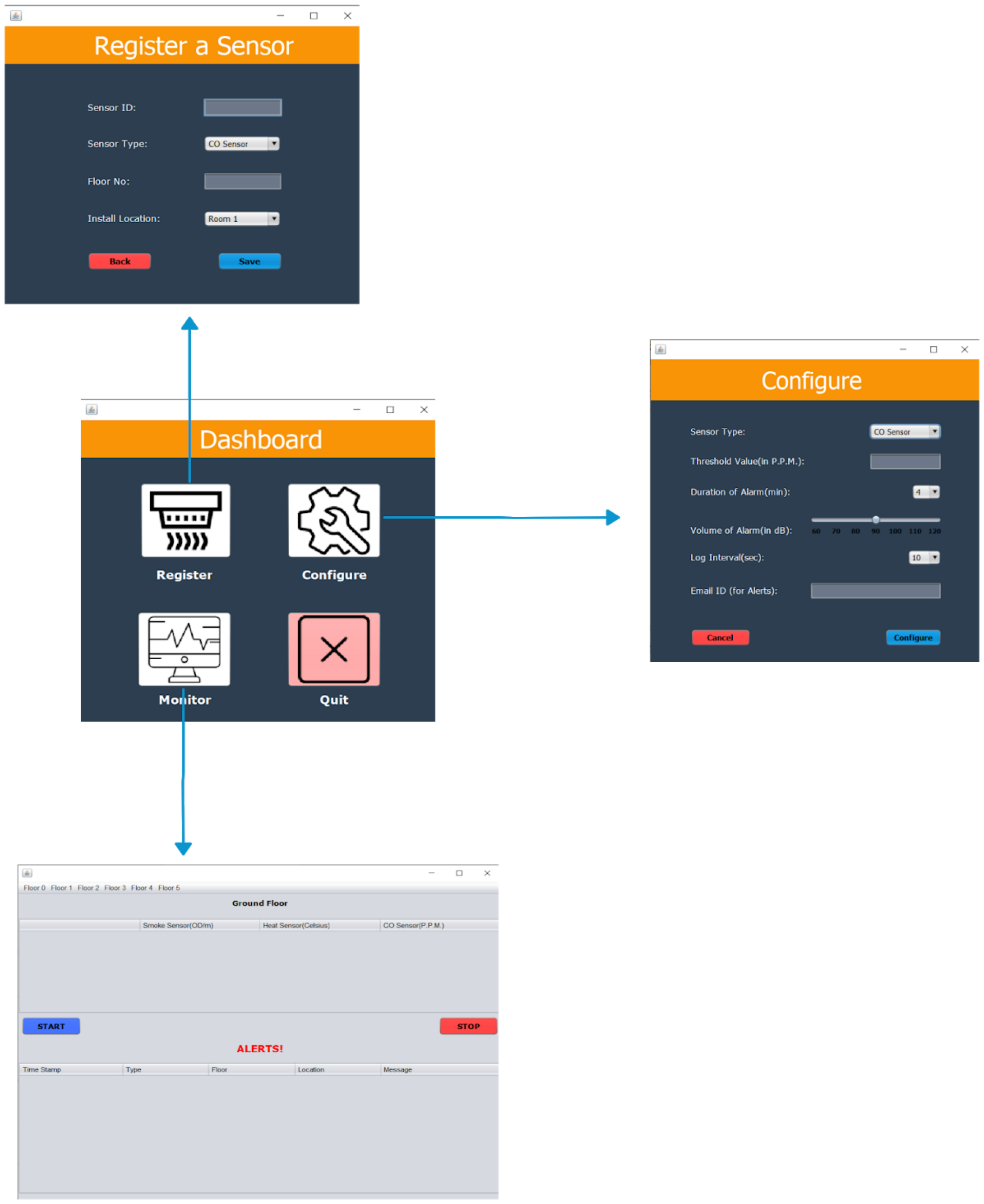


1. Then the Dashboard is displayed.



2. The Dashboard has 4 buttons corresponding to which it will display screens for further execution of the program.
   The execution is as follows:

3. After clicking on the Register button a new window is displayed in which a



Sensor ID:
Sensor Type: CO Sensor
Floor No:
Install Location: Room 1

Sensor Type: CO Sensor
Threshold Value(in P.P.M.):
Duration of Alarm(min): 4
Volume of Alarm(in dB): 60 70 80 90 100 110 120
Log Interval(sec): 10
Email ID (for Alerts):

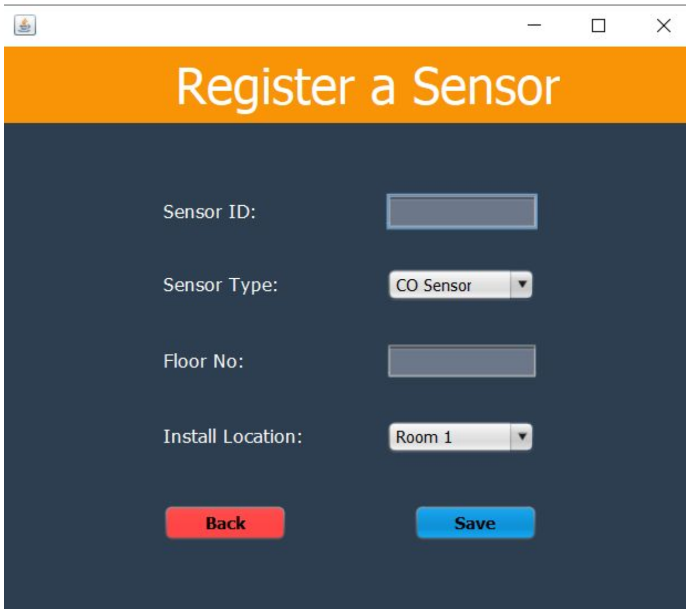
Floor 0 Floor 1 Floor 2 Floor 3 Floor 4 Floor 5

Ground Floor

	Smoke Sensor(OD/m)	Heat Sensor(Celsius)	CO Sensor(P.P.M.)

Time Stamp	Type	Floor	Location	Message

new sensor is registered.



Register a Sensor

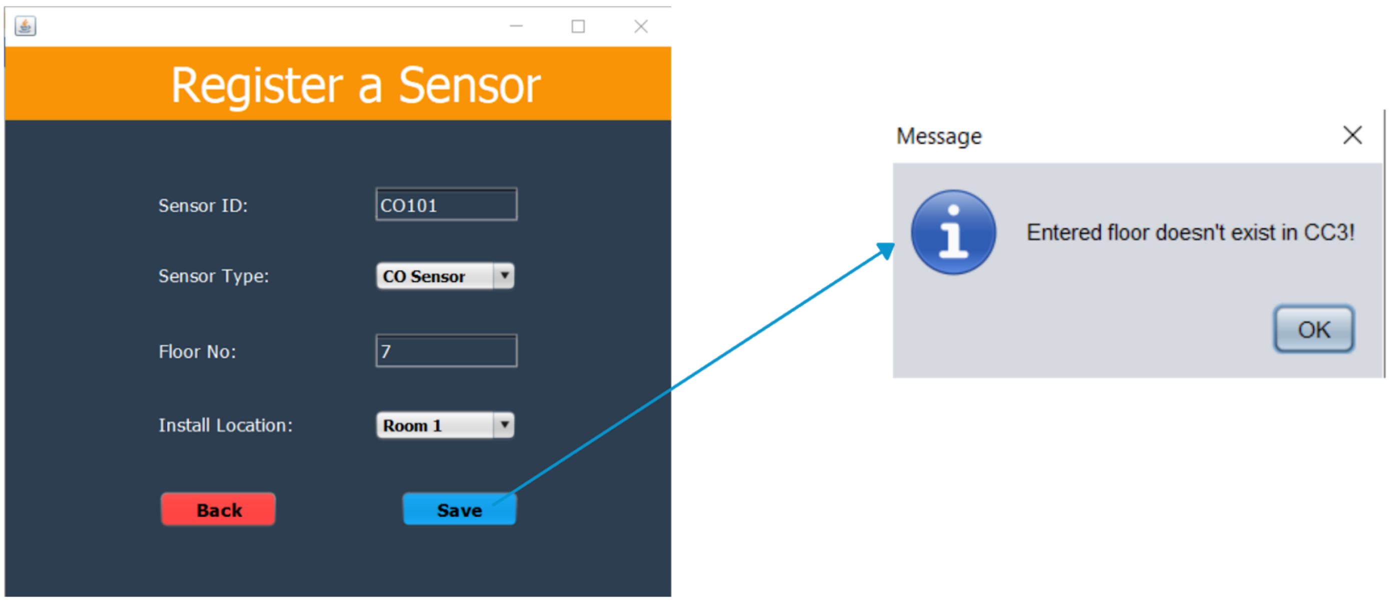
Sensor ID:

Sensor Type: CO Sensor

Floor No:

Install Location: Room 1

- The Sensor ID is a string.
- Sensor Type has three values namely: CO Sensor, Heat Sensor, Smoke Sensor.
- Floor no. can be from 0 to 5 and for Install Location we have Rooms 1 through 6, Lab 1, Lab 2 , Stairs 1, Stairs 2, and Hall.
- There is also validation that Sensor ID cannot be left blank and Floor No: must be from 0-5 otherwise it will give a popup error message.
- The Save button will save Sensor in memory and the Back button will direct it to MainScreen.



Register a Sensor

Sensor ID: CO101

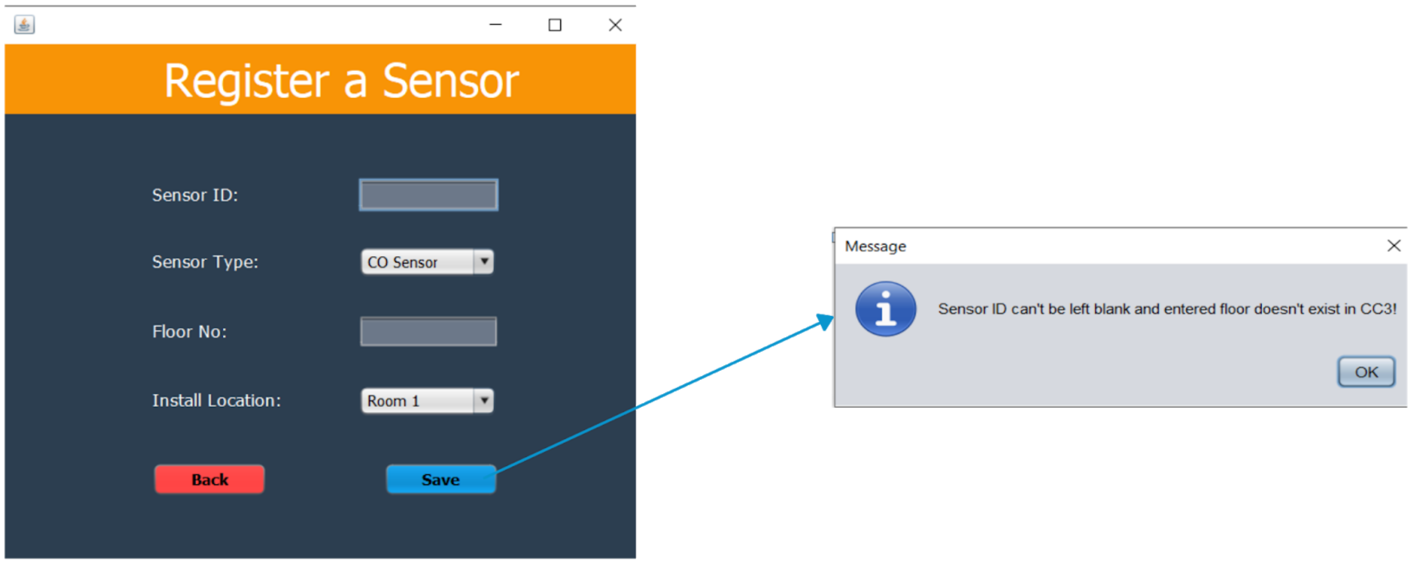
Sensor Type: CO Sensor

Floor No: 7

Install Location: Room 1

Message

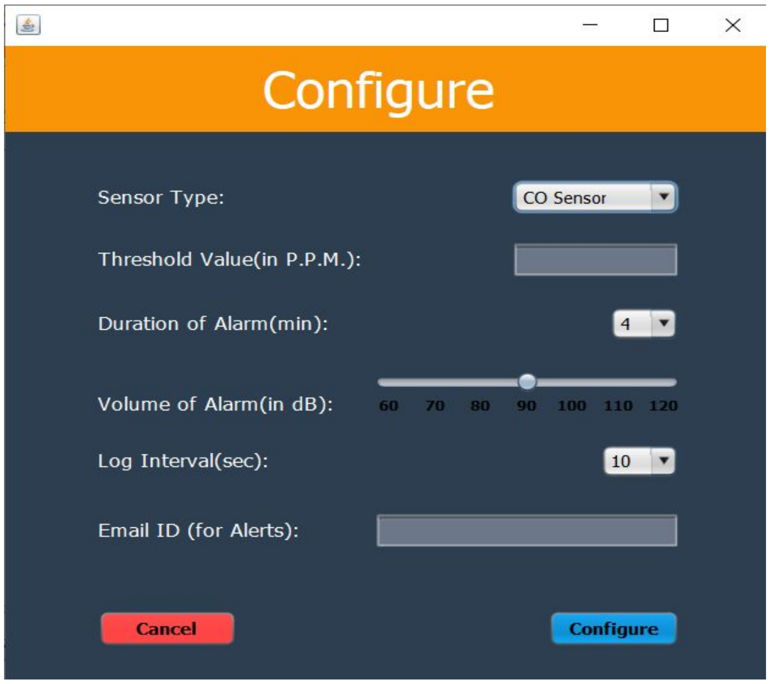
Entered floor doesn't exist in CC3!



Sensor ID can't be left blank and entered floor doesn't exist in CC3!

4. After registering sensors we have to configure them. For configuration we have a new window in which we can configure Threshold value, Duration of Alarm, Volume of Alarm, Log Interval and Email ID for a particular type of Sensor. The Threshold value of CO Sensor is in P.P.M, for Heat Sensor it is in degree celsius and for Smoke Sensor it is OD/metre. The Duration of Alarm can be 1,2,4 or 8 minutes. Volume of Alarm can range from 60 to 120 decibels. For Log Interval the value can be 10,15,30,60,120 seconds and Email ID for sending the Email if Alarm is triggered at some location.

   Threshold value must be a non-negative floating point no. for all three types and Email ID must have a format of x@y.z where x can contain alphabets,underscore and dot and y and z can be any string of alphabets otherwise it will give an error message. Threshold can't be left blank. Email ID can be left blank only if it has been entered while configuring for the first time. Also, the slider of volume and the combo boxes of log interval and will take the values according to the most recent configuration.



Configure

Sensor Type: CO Sensor

Threshold Value(in P.P.M.):

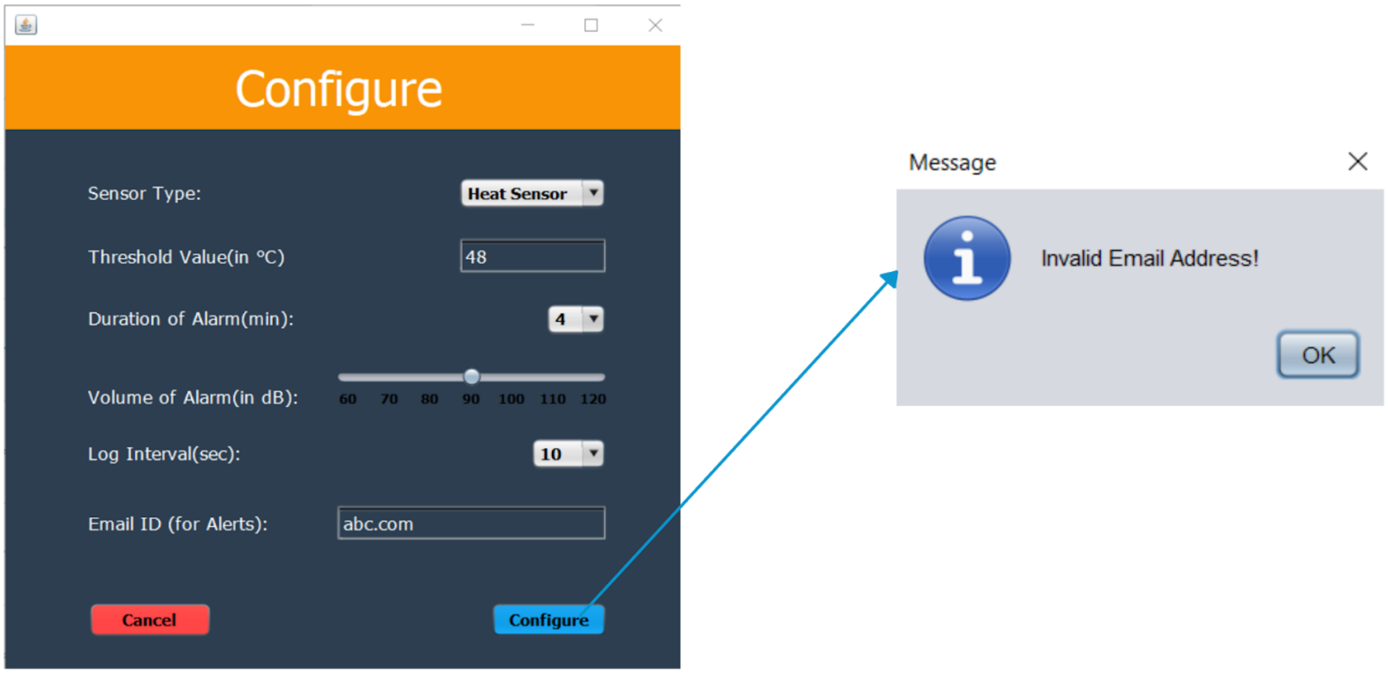
Duration of Alarm(min): 4

Volume of Alarm(in dB): 60 70 80 90 100 110 120

Log Interval(sec): 10

Email ID (for Alerts):

The Configure button will save it to memory and the Cancel button will direct to Dashboard.



Configure

Sensor Type: Heat Sensor

Threshold Value(in °C) 48

Duration of Alarm(min): 4

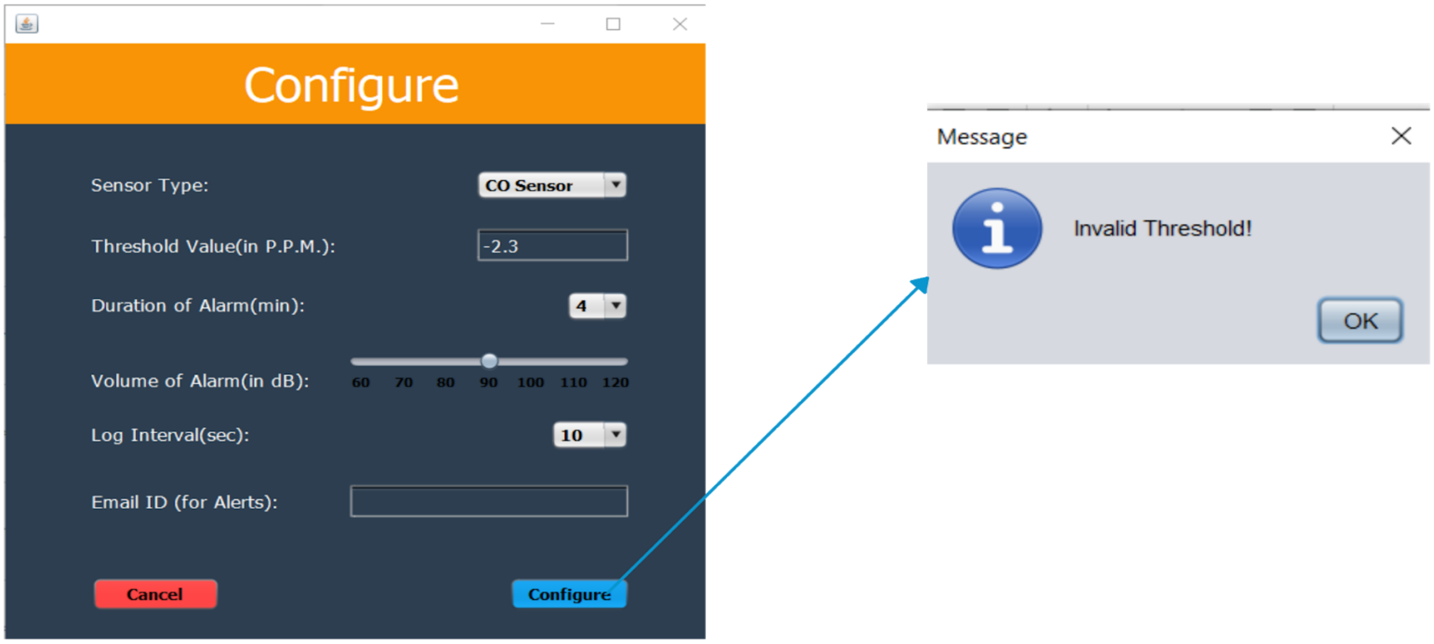
Volume of Alarm(in dB): 60 70 80 90 100 110 120

Log Interval(sec): 10

Email ID (for Alerts): abc.com

Message

Invalid Email Address!


Configure

Sensor Type: CO Sensor

Threshold Value(in P.P.M.): -2.3

Duration of Alarm(min): 4

Volume of Alarm(in dB): 60 70 80 90 100 110 120

Log Interval(sec): 10

Email ID (for Alerts):

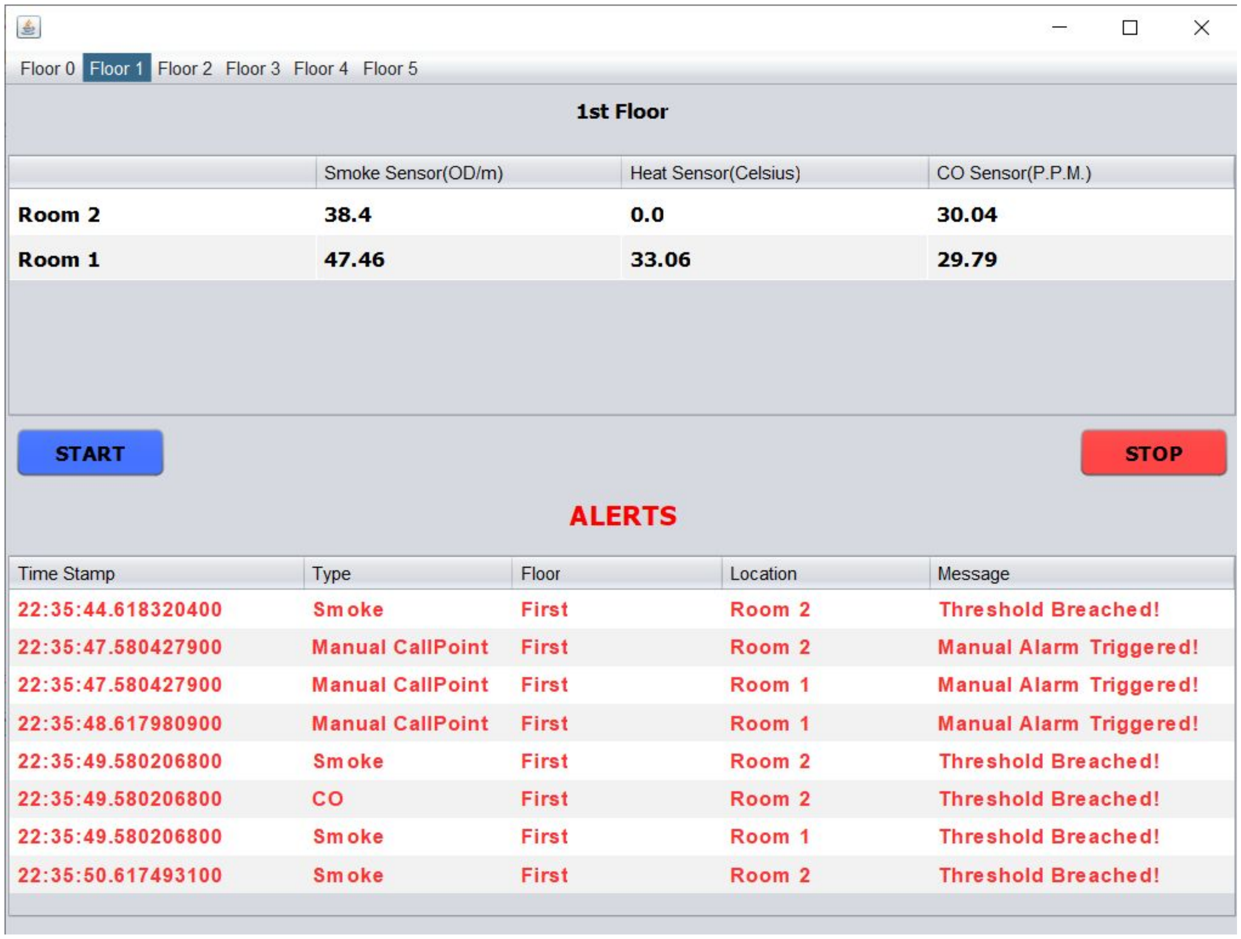
Cancel Configure

Message

Invalid Threshold!

OK

5. After registration and configuration, we can monitor the various sensors which are registered. For that we have the Monitoring Screen.



1st Floor

	Smoke Sensor(OD/m)	Heat Sensor(Celsius)	CO Sensor(P.P.M.)
Room 2	38.4	0.0	30.04
Room 1	47.46	33.06	29.79

ALERTS

Time Stamp	Type	Floor	Location	Message
22:35:44.618320400	Smoke	First	Room 2	Threshold Breached!
22:35:47.580427900	Manual CallPoint	First	Room 2	Manual Alarm Triggered!
22:35:47.580427900	Manual CallPoint	First	Room 1	Manual Alarm Triggered!
22:35:48.617980900	Manual CallPoint	First	Room 1	Manual Alarm Triggered!
22:35:49.580206800	Smoke	First	Room 2	Threshold Breached!
22:35:49.580206800	CO	First	Room 2	Threshold Breached!
22:35:49.580206800	Smoke	First	Room 1	Threshold Breached!
22:35:50.617493100	Smoke	First	Room 2	Threshold Breached!

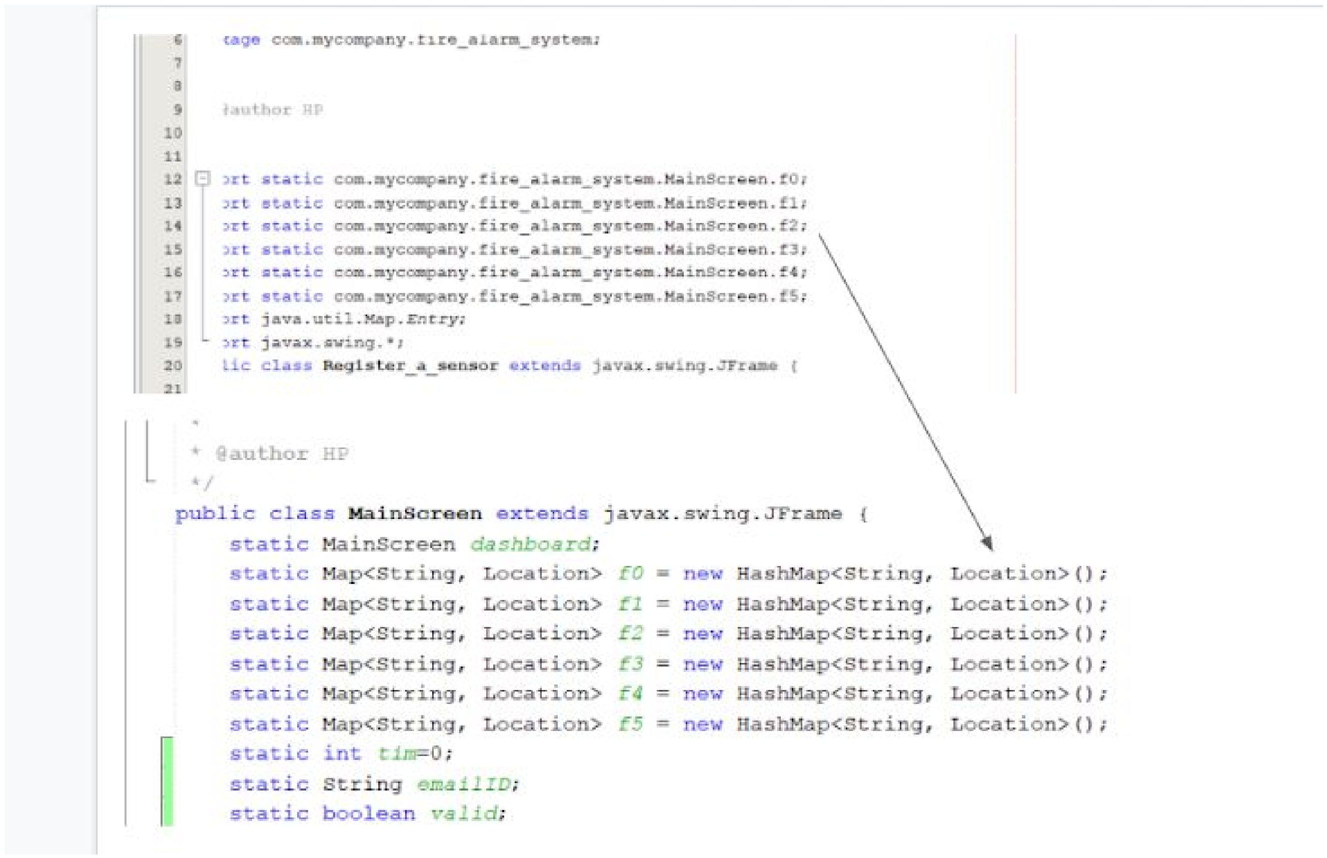
In the monitoring screen we can observe the value of all the sensors from Floor 0 to 5 which have been registered at a particular location. And if an alarm is triggered at a particular location it will be displayed in the Alerts table. The Screen is Dynamic with values changing in the interval of 2 seconds.
If the threshold is breached at any location an email will be sent within 30 seconds after the triggering of the alarm.

It has two buttons Start and Stop. Start will start the Monitoring and Stop is for directing to MainScreen.

6. In MainScreen we the Quit button for closing the program, it will cancel execution of all threads which are running in background and close the program.
   We also create a Log file which stores details of all necessary events which happen during the execution of the program. It logs data with date and time stamp when
   1) A new Sensor is registered.
   2) A particular type of Sensor is configured.
   3) An email is sent about the breach of Threshold.
   4) And values of all the registered sensors in the configured log interval.

BackEnd

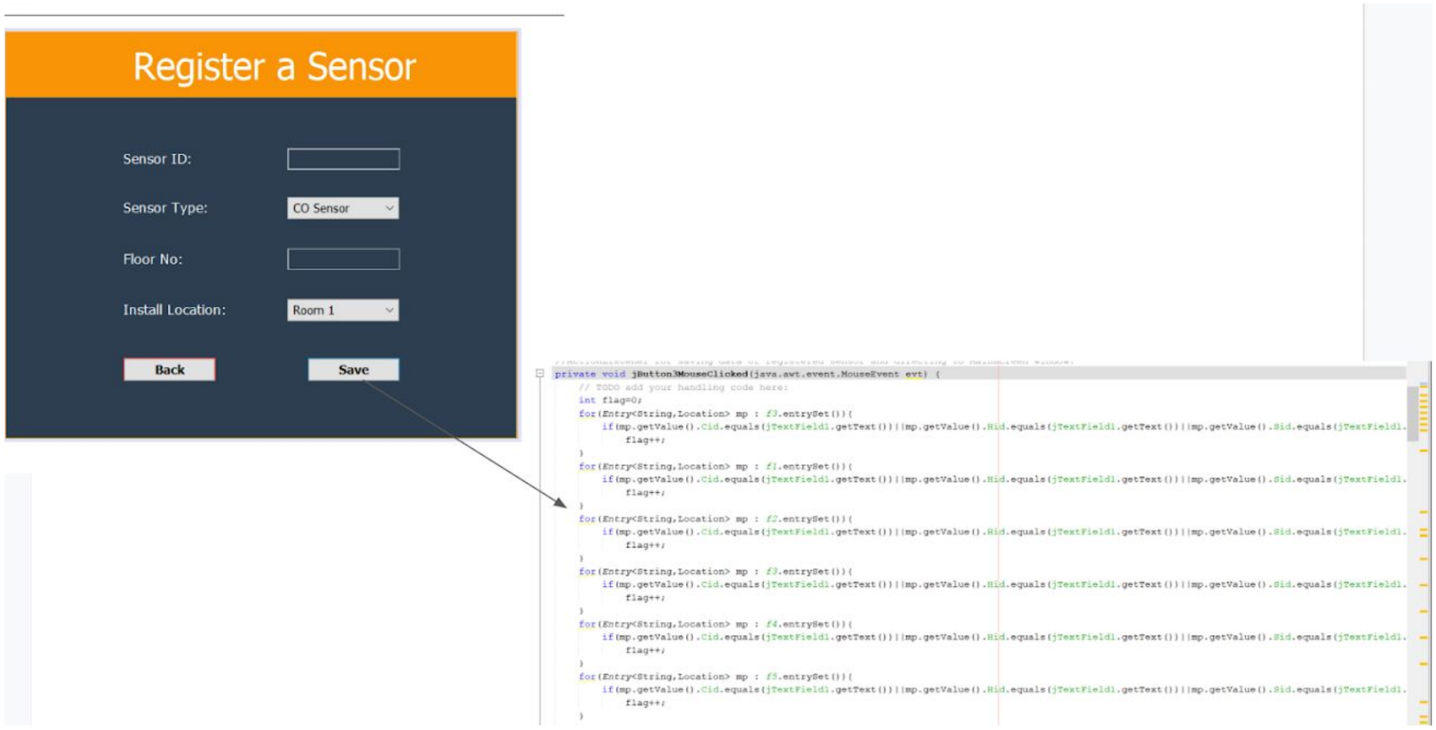
How and where are the various values of a sensor stored?



```
kage com.mycompany.fire_alarm_system;


 author HP


ort static com.mycompany.fire_alarm_system.MainScreen.f0;
ort static com.mycompany.fire_alarm_system.MainScreen.f1;
ort static com.mycompany.fire_alarm_system.MainScreen.f2;
ort static com.mycompany.fire_alarm_system.MainScreen.f3;
ort static com.mycompany.fire_alarm_system.MainScreen.f4;
ort static com.mycompany.fire_alarm_system.MainScreen.f5;
ort java.util.Map.Entry;
ort javax.swing.*;
lic class Register_a_sensor extends javax.swing.JFrame {
```

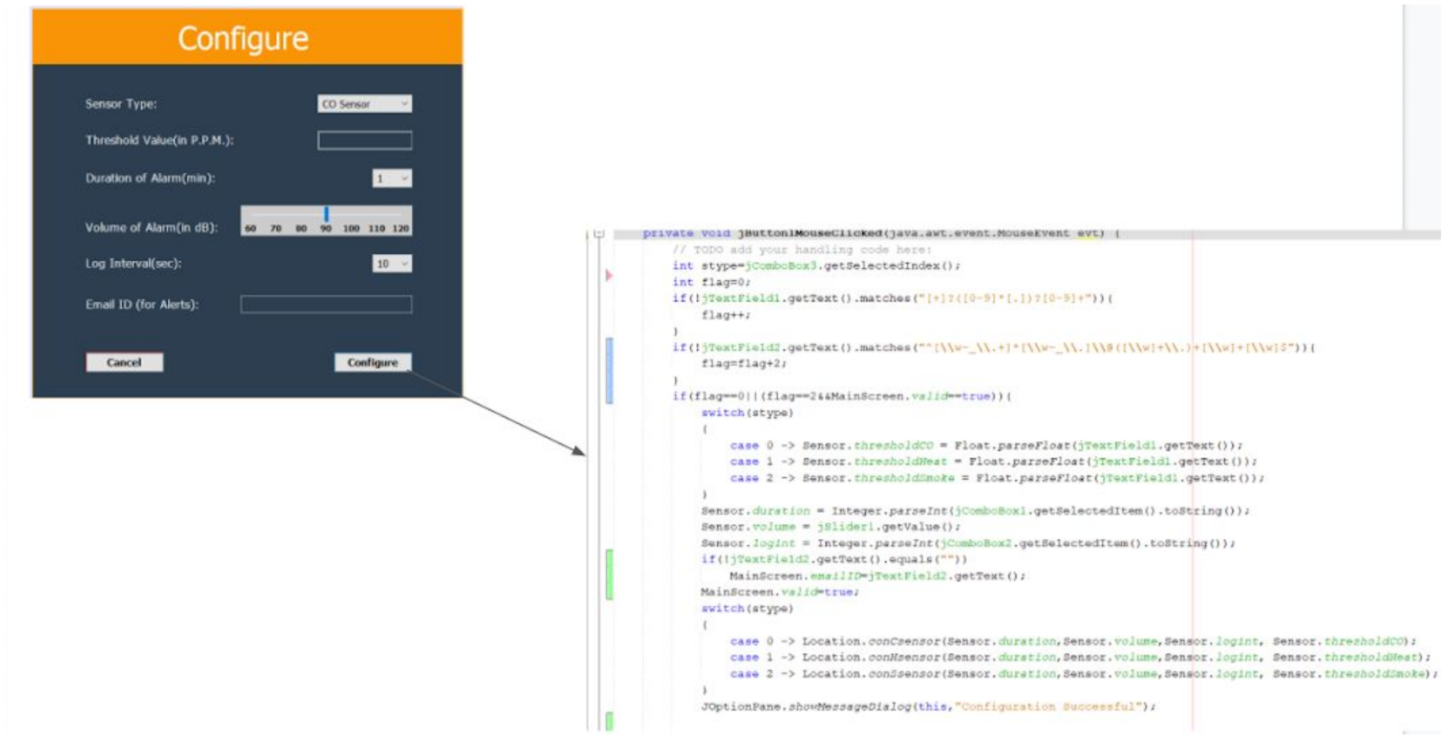
```
 *
 * @author HP
 */
public class MainScreen extends javax.swing.JFrame {
    static MainScreen dashboard;
    static Map<String, Location> f0 = new HashMap<String, Location>();
    static Map<String, Location> f1 = new HashMap<String, Location>();
    static Map<String, Location> f2 = new HashMap<String, Location>();
    static Map<String, Location> f3 = new HashMap<String, Location>();
    static Map<String, Location> f4 = new HashMap<String, Location>();
    static Map<String, Location> f5 = new HashMap<String, Location>();
    static int tim=0;
    static String emailID;
    static boolean valid;
```

The 1st part of the image is of Register_a_Sensor class; the line at the base of the arrow points to a hashmap defined for each of the floors in which the data of the sensors of the respective floor will be stored ; the code snippet at the tip of the arrow .



```
private void jButton3MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int flag=0;
    for(Entry<String,Location> mp : f3.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
    for(Entry<String,Location> mp : f1.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
    for(Entry<String,Location> mp : f2.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
    for(Entry<String,Location> mp : f3.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
    for(Entry<String,Location> mp : f4.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
    for(Entry<String,Location> mp : f5.entrySet()){
        if(mp.getValue().Cid.equals(jTextField1.getText())||mp.getValue().Hid.equals(jTextField1.getText())||mp.getValue().Sid.equals(jTextField1.
            flag++;
    }
```

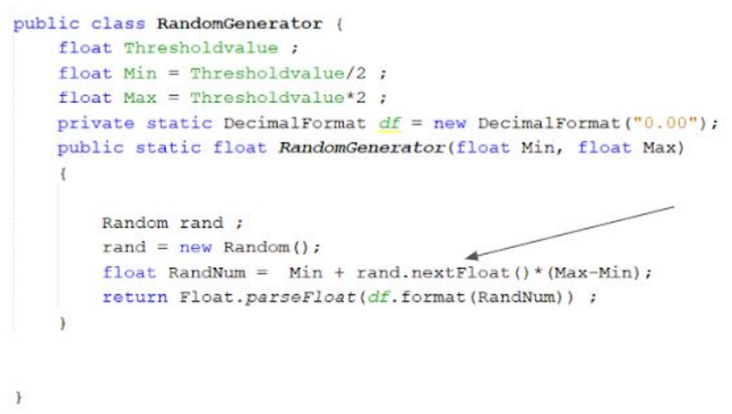
The above image shows the logic of how values are saved. The save button points to the code snippet which shows the values of sensors being stored in the maps of their respective floors.



```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int stype=jComboBox3.getSelectedIndex();
    int flag=0;
    if(!jTextField1.getText().matches("[+]?([0-9]*[.])?[0-9]+")){
        flag++;
    }
    if(!jTextField2.getText().matches("^[\\w-_\\.+]*[\\w-_\\.]\\@([\\w]+\\.)+[\\w]+[\\w]$")){
        flag=flag+2;
    }
    if(flag==0||(flag==2&&MainScreen.valid==true)){
        switch(stype)
        {
            case 0 -> Sensor.thresholdCO = Float.parseFloat(jTextField1.getText());
            case 1 -> Sensor.thresholdHeat = Float.parseFloat(jTextField1.getText());
            case 2 -> Sensor.thresholdSmoke = Float.parseFloat(jTextField1.getText());
        }
        Sensor.duration = Integer.parseInt(jComboBox1.getSelectedItem().toString());
        Sensor.volume = jSlider1.getValue();
        Sensor.logint = Integer.parseInt(jComboBox2.getSelectedItem().toString());
        if(!jTextField2.getText().equals(""))
            MainScreen.emailID=jTextField2.getText();
        MainScreen.valid=true;
        switch(stype)
        {
            case 0 -> Location.conCsensor(Sensor.duration,Sensor.volume,Sensor.logint, Sensor.thresholdCO);
            case 1 -> Location.conHsensor(Sensor.duration,Sensor.volume,Sensor.logint, Sensor.thresholdHeat);
            case 2 -> Location.conSsensor(Sensor.duration,Sensor.volume,Sensor.logint, Sensor.thresholdSmoke);
        }
        JOptionPane.showMessageDialog(this,"Configuration Successful");
```

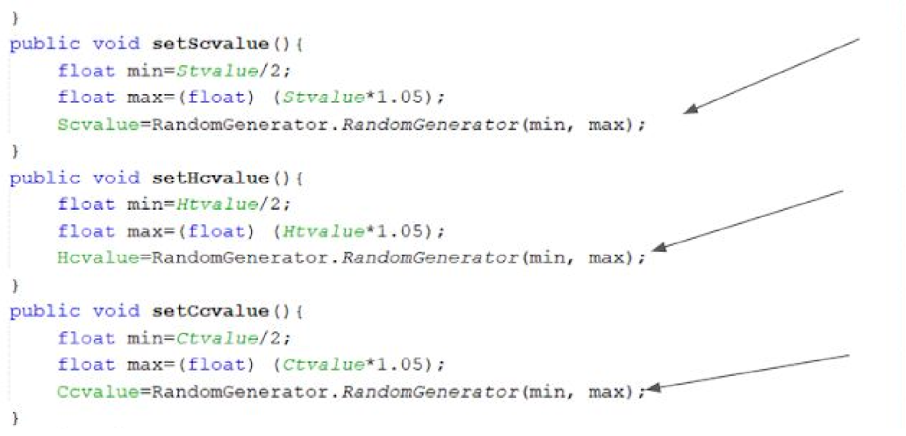
The above image shows how the configuration values are registered. The configure button points to the code snippet which stores the values using conCsensor, conSsensor and conHsensor .

How is the value of a sensor randomised?



```
public class RandomGenerator {
    float Thresholdvalue ;
    float Min = Thresholdvalue/2 ;
    float Max = Thresholdvalue*2 ;
    private static DecimalFormat df = new DecimalFormat("0.00");
    public static float RandomGenerator(float Min, float Max)
    {

        Random rand ;
        rand = new Random();
        float RandNum =  Min + rand.nextFloat()*(Max-Min);
        return Float.parseFloat(df.format(RandNum)) ;
    }


}
```

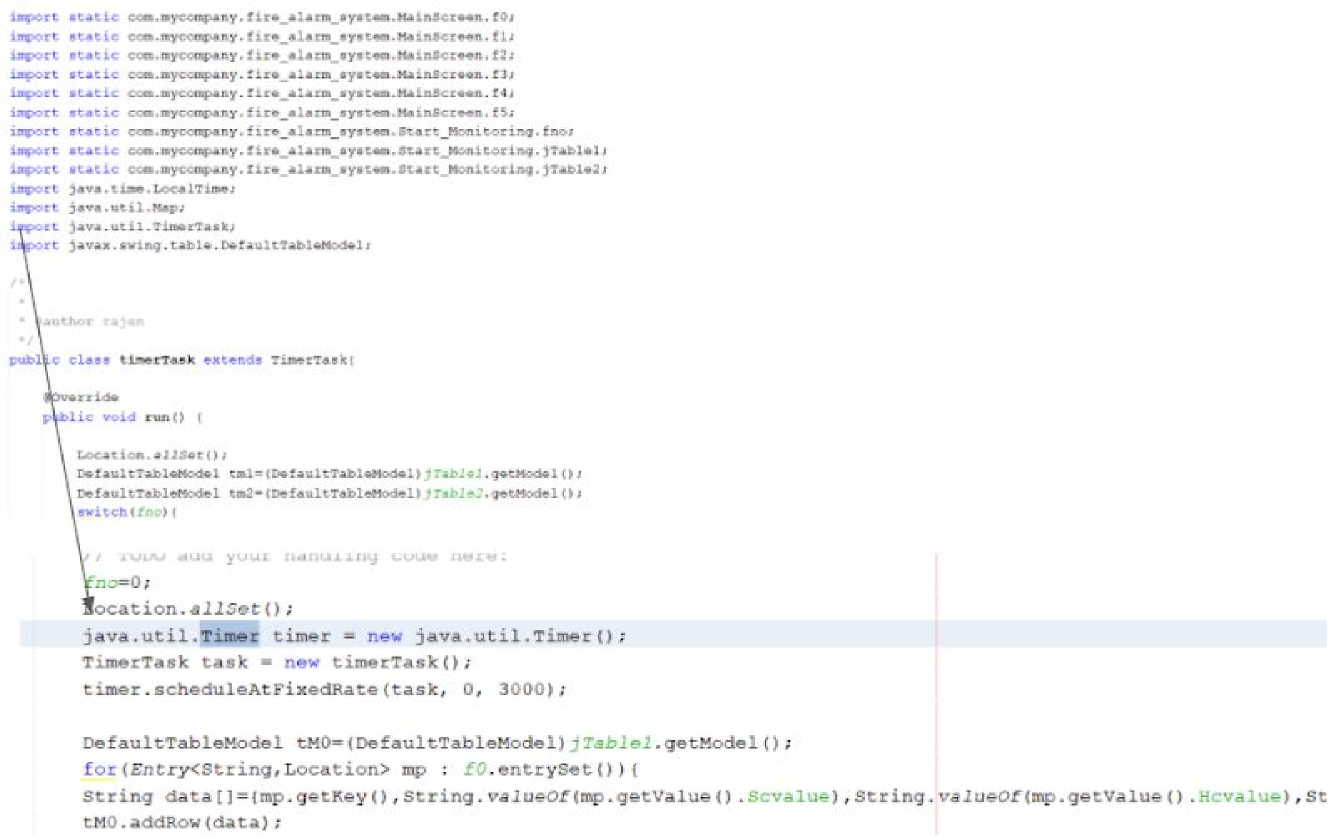
This is the code snippet of RandomGenerator class which generates random values between Min and Max. This class uses the java.util.Random class to execute this . The arrow points to the execution of rand.nextFloat() to print random numbers .



```
}
public void setScvalue(){
    float min=Stvalue/2;
    float max=(float) (Stvalue*1.05);
    Scvalue=RandomGenerator.RandomGenerator(min, max);
}
public void setHcvalue(){
    float min=Htvalue/2;
    float max=(float) (Htvalue*1.05);
    Hcvalue=RandomGenerator.RandomGenerator(min, max);
}
public void setCcvalue(){
    float min=Ctvalue/2;
    float max=(float) (Ctvalue*1.05);
    Ccvalue=RandomGenerator.RandomGenerator(min, max);
}
```

This is a code snippet of Location class .
The arrows show the implementation of RandomGenerator class.

How is the value updated at a regular interval ?

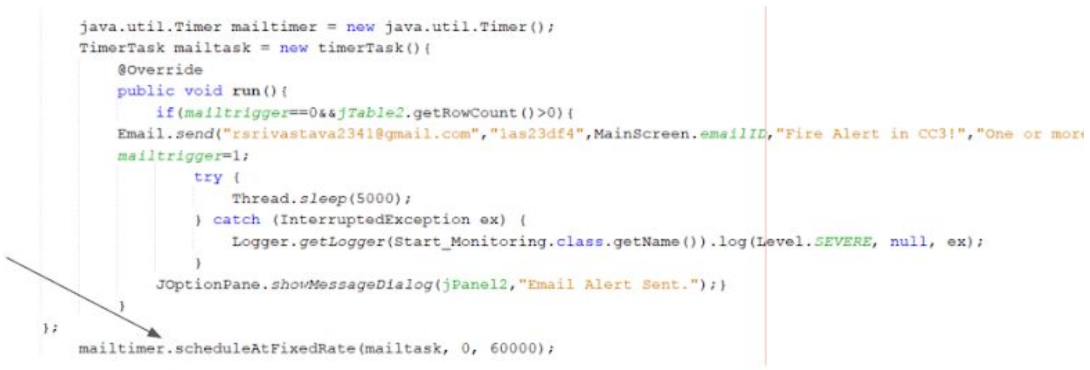


```
import static com.mycompany.fire_alarm_system.MainScreen.f0;
import static com.mycompany.fire_alarm_system.MainScreen.f1;
import static com.mycompany.fire_alarm_system.MainScreen.f2;
import static com.mycompany.fire_alarm_system.MainScreen.f3;
import static com.mycompany.fire_alarm_system.MainScreen.f4;
import static com.mycompany.fire_alarm_system.MainScreen.f5;
import static com.mycompany.fire_alarm_system.Start_Monitoring.fno;
import static com.mycompany.fire_alarm_system.Start_Monitoring.jTable1;
import static com.mycompany.fire_alarm_system.Start_Monitoring.jTable2;
import java.time.LocalTime;
import java.util.Map;
import java.util.TimerTask;
import javax.swing.table.DefaultTableModel;

/*
 *
 * @author rajan
 */
public class timerTask extends TimerTask{

    @Override
    public void run() {

        Location.allSet();
        DefaultTableModel tm1=(DefaultTableModel)jTable1.getModel();
        DefaultTableModel tm2=(DefaultTableModel)jTable2.getModel();
        switch(fno){
```

```
        // TODO add your handling code here:
        fno=0;
        Location.allSet();
        java.util.Timer timer = new java.util.Timer();
        TimerTask task = new timerTask();
        timer.scheduleAtFixedRate(task, 0, 3000);

        DefaultTableModel tM0=(DefaultTableModel)jTable1.getModel();
        for(Entry<String,Location> mp : f0.entrySet()){
        String data[]={mp.getKey(),String.valueOf(mp.getValue().Scvalue),String.valueOf(mp.getValue().Hcvalue),St
        tM0.addRow(data);
```

The 1st part of the above image shows the code snippet of timerTask class.This class uses the classes java.util.Timer and java.util.TimerTask to schedule processes . The 2nd part of the image shows the snippet of the Start Monitoring class, which shows the implementation of timerTask class.

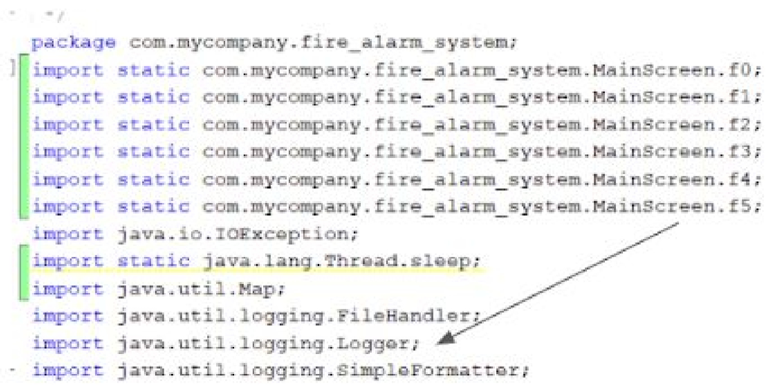
The timer.scheduleAtFixedRate method will perform “task” at a regular interval of 2 seconds .



```
    java.util.Timer mailtimer = new java.util.Timer();
    TimerTask mailtask = new timerTask(){
        @Override
        public void run(){
            if(mailtrigger==0&&jTable2.getRowCount()>0){
        Email.send("rsrivastava2341@gmail.com","1as23df4",MainScreen.emailID,"Fire Alert in CC3!","One or mor
        mailtrigger=1;
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Start_Monitoring.class.getName()).log(Level.SEVERE, null, ex);
                }
            JOptionPane.showMessageDialog(jPanel2,"Email Alert Sent.");}
        }
};
    mailtimer.scheduleAtFixedRate(mailtask, 0, 60000);
```

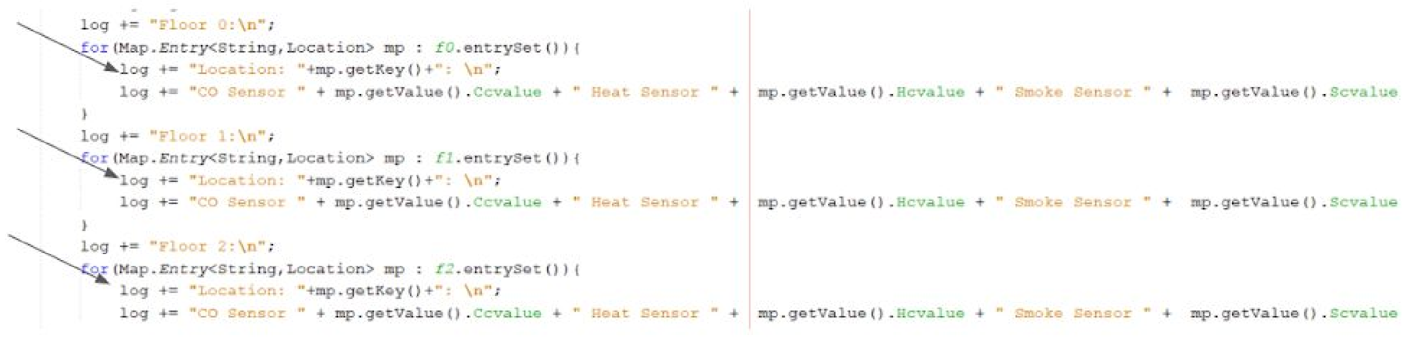
The above snippet shows the implementation of timerTask class in sending email using mailtimer.scheduleAtFixedRate method.

Log generation



```java
package com.mycompany.fire_alarm_system;
import static com.mycompany.fire_alarm_system.MainScreen.f0;
import static com.mycompany.fire_alarm_system.MainScreen.f1;
import static com.mycompany.fire_alarm_system.MainScreen.f2;
import static com.mycompany.fire_alarm_system.MainScreen.f3;
import static com.mycompany.fire_alarm_system.MainScreen.f4;
import static com.mycompany.fire_alarm_system.MainScreen.f5;
import java.io.IOException;
import static java.lang.Thread.sleep;
import java.util.Map;
import java.util.logging.FileHandler;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
```

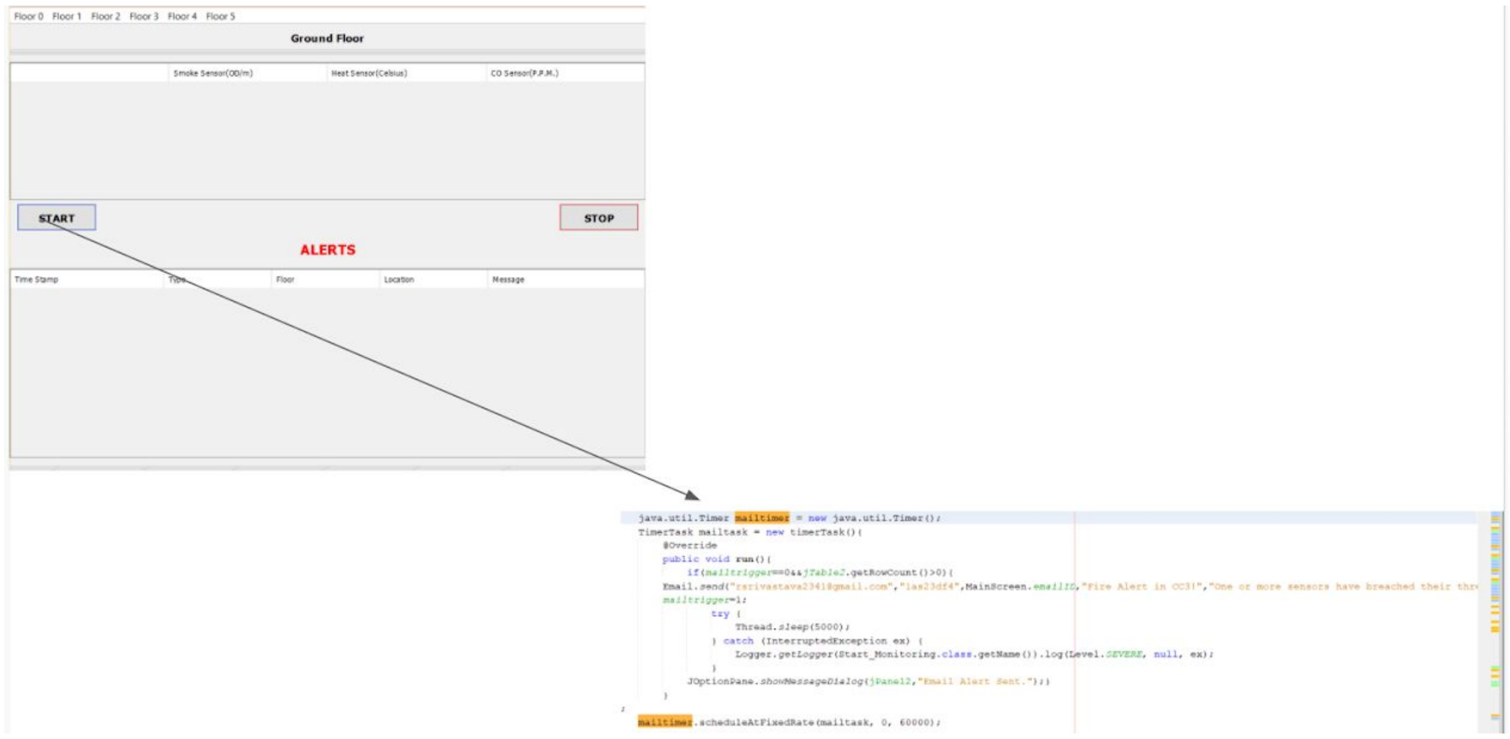
The above snippet is of the LogGenerator class which generates a log report of the registered sensors in the building. The main classes used are of class java.util.logging and class java.lang.Thread.sleep .



```java
log += "Floor 0:\n";
for(Map.Entry<String,Location> mp : f0.entrySet()){
    log += "Location: "+mp.getKey()+": \n";
    log += "CO Sensor " + mp.getValue().Ccvalue + " Heat Sensor " +  mp.getValue().Hcvalue + " Smoke Sensor " +  mp.getValue().Scvalue
}
log += "Floor 1:\n";
for(Map.Entry<String,Location> mp : f1.entrySet()){
    log += "Location: "+mp.getKey()+": \n";
    log += "CO Sensor " + mp.getValue().Ccvalue + " Heat Sensor " +  mp.getValue().Hcvalue + " Smoke Sensor " +  mp.getValue().Scvalue
}
log += "Floor 2:\n";
for(Map.Entry<String,Location> mp : f2.entrySet()){
    log += "Location: "+mp.getKey()+": \n";
    log += "CO Sensor " + mp.getValue().Ccvalue + " Heat Sensor " +  mp.getValue().Hcvalue + " Smoke Sensor " +  mp.getValue().Scvalue
```

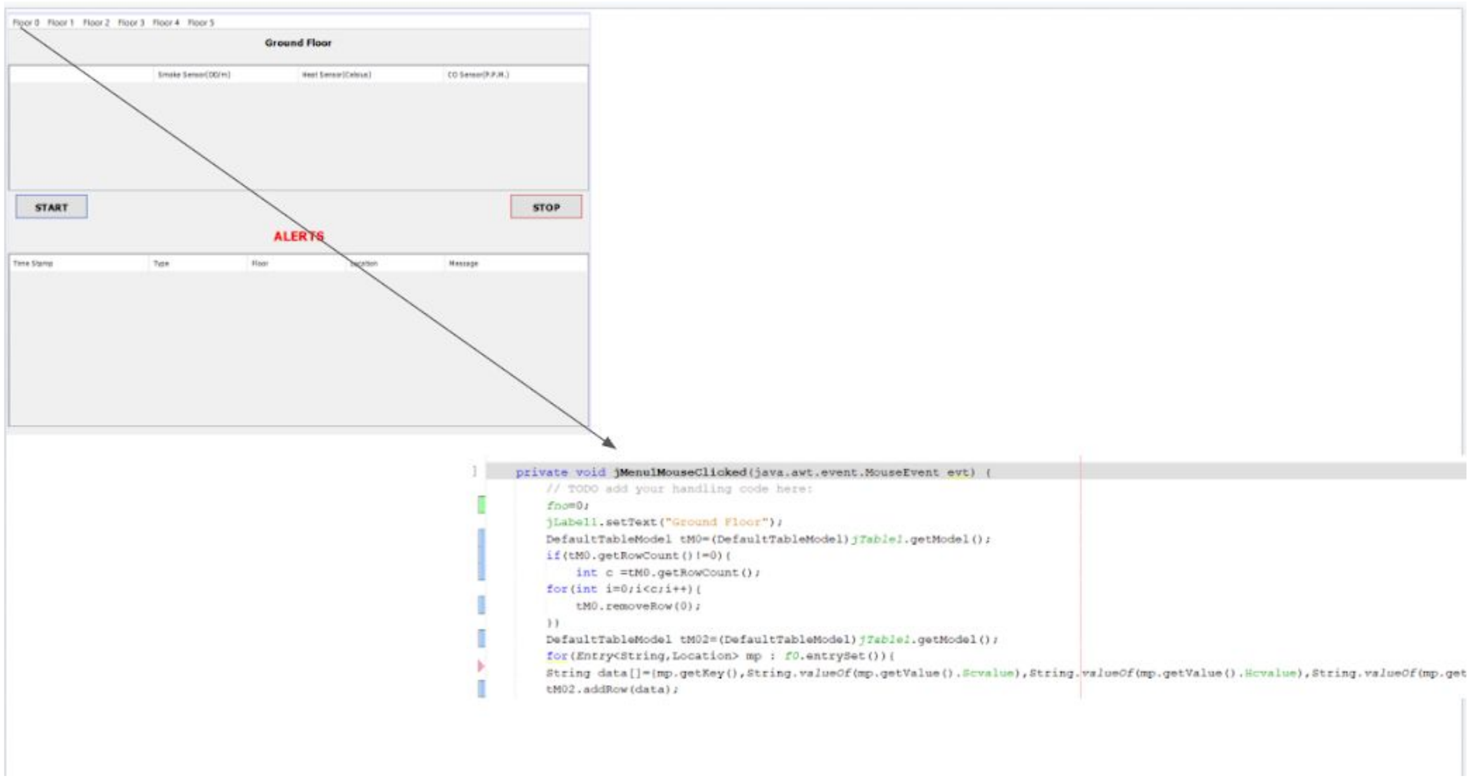
The arrowheads point to the entry of sensor data into log .

Data Generation in Start_Monitoring class



```
java.util.Timer mailtimer = new java.util.Timer();
TimerTask mailtask = new timerTask(){
    @Override
    public void run(){
        if(mailtrigger==0&&jTable2.getRowCount()>0){
    Email.send("rsrivastava2341@gmail.com","1as23df4",MainScreen.emailID,"Fire Alert in CC3!","One or more sensors have breached their thr
    mailtrigger=1;
            try {
                Thread.sleep(5000);
            } catch (InterruptedException ex) {
                Logger.getLogger(Start_Monitoring.class.getName()).log(Level.SEVERE, null, ex);
            }
        JOptionPane.showMessageDialog(jPanel2,"Email Alert Sent.");}
    }
;
mailtimer.scheduleAtFixedRate(mailtask, 0, 60000);
```

The base of the arrow is at the Start button, which when clicked , implements the code snippet pointed by the arrow. This code snippet creates a new timertask , stores the data of sensors in the table and repeats this process every 3 seconds , and in case of threshold breach , sends an email to the registered email ID after a fixed interval of 60 seconds .



```java
    private void jMenu1MouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        fno=0;
        jLabel1.setText("Ground Floor");
        DefaultTableModel tM0=(DefaultTableModel)jTable1.getModel();
        if(tM0.getRowCount()!=0){
            int c =tM0.getRowCount();
        for(int i=0;i<c;i++){
            tM0.removeRow(0);
        }}
        DefaultTableModel tM02=(DefaultTableModel)jTable1.getModel();
        for(Entry<String,Location> mp : f0.entrySet()){
        String data[]={mp.getKey(),String.valueOf(mp.getValue().Scvalue),String.valueOf(mp.getValue().Hcvalue),String.valueOf(mp.get
        tM02.addRow(data);
```

In the above image the arrow base points to a menu which shows sensor values of registered sensors at the ground floor, implementing the code snippet at the arrowhead . The same is for other floors .

Conclusion:

This is a comprehensive solution for the management of smart fire alarm systems which can be implemented with minimal modification. Possible future improvements can be added as shown below.

Future Improvements:

- Extend the program to edit and add locations and floors.
- Implement alerts in accordance with the difference between the threshold and normal conditions(according to seasons) instead of absolute threshold.
- Close alerts if Alarm duration is crossed.
- Modify Random Generator to generate values in a pattern.

References

- https://www.vogella.com/tutorials/Logging/article.html
- https://www.javatpoint.com/example-of-sending-email-using-java-mail-api-through-gmail-server
- https://www.geeksforgeeks.org/logging-in-java/
- https://www.stackexchange.com/
- https://crunchify.com/simplest-way-to-generate-logs-in-java-using-java-util-loggings-simpleformatter-and-xmlformatter/
- https://javarevisited.blogspot.com/2013/02/what-is-timer-and-timertask-in-java-example-tutorial.html#axzz6drwagdY0
- https://stackoverflow.com/questions/3179136/jtable-how-to-refresh-table-model-after-insert-delete-or-update-the-data