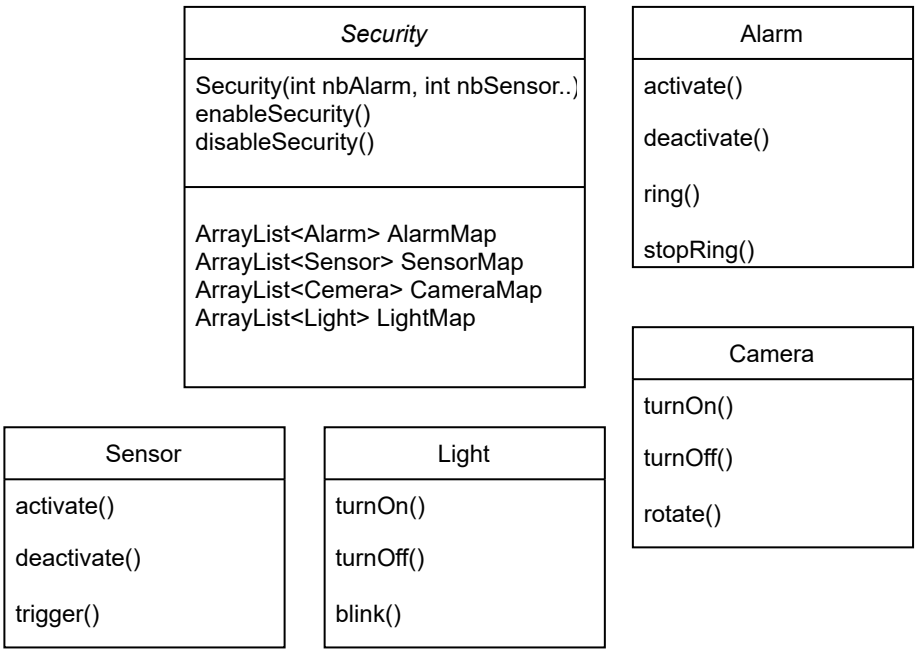
TP3 - Facade et Singleton

#DesignPattern

Facade

La façade à pour objectif de fournir une inteface inifiée et de plus haut niveau à un ensemble d'interfaces d'un sous-système, afin de le rendre plus facile à utiliser et ce afin de masquer un sous-ensemble complexe dans une interface simple.



Cela donnerait en code :

```
public class SecurityFacade {
    ArrayList<Sensor> SensorMap = new ArrayList<Sensor>();
    ArrayList<Camera> CameraMap = new ArrayList<Camera>();
    ArrayList<Light> LightMap = new ArrayList<Light>();
    ArrayList<Alarm> AlarmMap = new ArrayList<Alarm>();

    public SecurityFacade(int nbSensors, int nbCameras, int nbLights,
int nbAlarms) {
        for (int i = 0; i < nbSensors; i++) {
            SensorMap.add(new Sensor());
        }
        for (int i = 0; i < nbCameras; i++) {
```

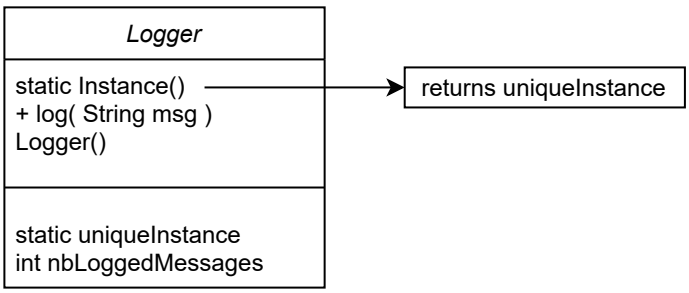
```
            CameraMap.add(new Camera());
        }
        for (int i = 0; i < nbLights; i++) {
            LightMap.add(new Light());
        }
        for (int i = 0; i < nbAlarms; i++) {
            AlarmMap.add(new Alarm());
        }
    .....
    protected void enableSecurity() {
        for (Camera camera : CameraMap) {
                    camera.turnOn();
                }

                for (Sensor sensor : SensorMap) {
                    sensor.activate();
                }
                for (Light light : LightMap) {
                    light.turnOff();
                }
                for (Alarm alarm : AlarmMap) {
                    alarm.activate();
            }
        }
    }
// pareil pour disableSecurity mais dans l'autre sens
}

/*****************************************************************************

public class Light {

    public void turnOn() {
        System.out.println("La lumière est allumée.");
    }

    public void turnOff() {
        System.out.println("La lumière est éteinte.");
    }

    public void blink() {
        System.out.println("La lumière clignote.");
```

```
    }

// pareil pour les autres éléments du système cf : Alarm, Camera, etc.
/*****************************************************************************

public class Client {
    private SecurityFacade mySafetySystem;

    public Client() {
    this.mySafetySystem = new SecurityFacade(10, 10, 10, 10);
    }

    public SecurityFacade getMySafetySystem() {

        return this.mySafetySystem;
    }

    public void enableSecurity() {
        mySafetySystem.enableSecurity();
    }

    public void disableSecurity() {
        mySafetySystem.disableSecurity();
    }
}
```

- SecurityFacade regroupe toutes les méthodes qui permettent de manipuler les éléments du sous-système simplement, sait quelles classes du sous-système sont responsables de telle ou telle requête et délègue le traitement des requêtes du client aux objets appropriés du sous-système.
- Alarm,Camera,Sensor, Light, etc... implémentent les fonctionnalités du sous-système traitent les requêtes émises par la façade mais ne connaissent pas cette dernière (pas de référence.)

Singleton

L'objectif du singleton est de garantir qu'une classe n'existe qu'une fois et permet de fournir un point d'accès global à cette instance sans pour autant créer une variable globale.




```java
// le logger sera notre unique instance.
public class Logger {
    private int nbLoggedMessages;
    private static Logger uniqueInstance = null;

    public static Logger Instance() {
        if(uniqueInstance == null) {
        uniqueInstance = new Logger();
        }
        return( uniqueInstance );
    }

    private Logger() {
        nbLoggedMessages = 0;
    }

    public void log( String msg ) {
        this.nbLoggedMessages++;
        System.out.println("LOG" + this.nbLoggedMessages + ": " + msg);
    }
}

/*****************************************************************************


public class Main {


    public static void main(String[] args) {
        Logger loggy = Logger.Instance();
        loggy.log("oui");
        loggy.log("non");
        Logger loggyoui = Logger.Instance();
    }
}
```

- Logger construit sa propre instance unique et définit une méthode de classe Instance() qui donne l'accès à son unique instance.
- Main accède à l'instance de Logger via la méthode de classe Instance();