TD – Patron de conception “Observer”

Objectifs du TD : illustrer l’utilisation du patron de conception (design pattern) Observer

1. Entité : Observable (Subject) sans le patron

Description de l’exercice : Le but de cet exercice est de simuler le fonctionnement d’un capteur de présence qui utilise une caméra. Le capteur de présence déclenche une détection si le pourcentage des pixels modifiés d’une image à une autre dépasse un seuil

Définir la classe **`CapteurPresence`** qui a pour :

- Attributs :
  - **`seuil`** : entier passé en paramètre lors de l’instanciation de la classe.
  - **`pourcentage`** : un entier initialisé à **`0`** lors de l’instanciation de la classe.
- Méthodes :
  - **`traiterImage`**`()` : qui met à jour l’attribut pourcentage avec une valeur aléatoire entre **`0`** et **`100`**. (Utiliser la fonction `random.randint(a, b)` pour générer un nombre aléatoire dans `[a, b]`)
  - **`detecterMouvement`**`()` : qui teste la valeur de l’attribut **`pourcentage`** à celle de **`seuil`**. Le résultat de la détection est retourné par la méthode.

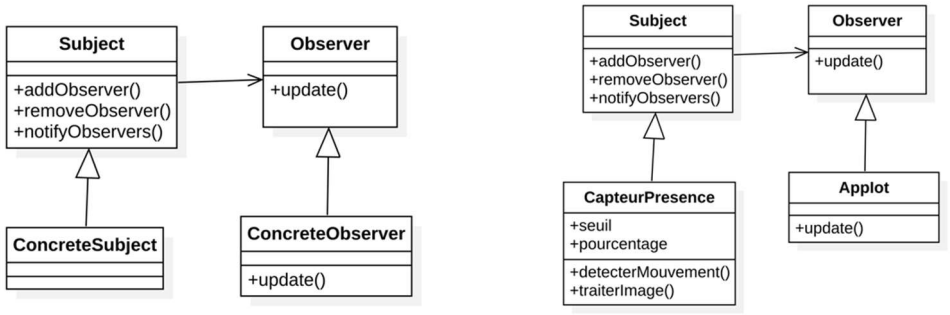
Ecrire un programme principal qui

- Instancie un objet `capPresence` de la classe `CapteurPresence` avec un seuil de détection à `30%`
- Exécute une boucle infinie qui :
  - Traite une image
  - Détecte s’il y a un mouvement et affiche un message «`Mouvement détecté !`» si le résultat de `detecterMouvement` est `VRAI`.
  - Fait une pause de 2 secondes

2. Entité : Observateur (avec le patron « observer »)

Description de l’exercice : Le but de cet exercice est d’écrire un programme pour exploiter le capteur de présence défini dans l’exercice 1 avec le patron « observer ».

Le patron est donné ci-dessous , ainsi que le code la classe « Subject ». Le ConcreteSubject devenant la classe CapteurPresence, il est demandé de créer une classe AppIot qui remplacera la classe ConcreteObserver.



Vous aurez à programmer les 4 classes du diagramme et à réaliser le programme principal.
Le code de Subject vous est donné ci-dessous :

```
class Subject:
    def __init__(self):
        self._observers = []

    def add_observer(self, observer):
        self._observers.append(observer)

    def remove_observer(self, observer):
        self._observers.remove(observer)

    def notify_observers(self, presence_detected):
        for observer in self._observers:
            observer.update(presence_detected)
```

1) Ecrire la classe AppIoT qui a pour principale méthode update() qui affiche un message simple « Présence détectée ». Ecrire la classe Observer ne comportant que la méthode vide update()
2) Modifiez la classe CapteurPrésence pour qu'elle hérite de Subject, utilise le constructeur de Subject, et appelle la méthode notify quand une présence est détectée.
3) Ecrire le diagramme de séquence et le programme principal qui créé un objet CapteurPresence, AppIot, Ajoute l'applot au capteur en tant qu'observateur, et effectue en boucle le traitement de l'image et la detection.

3. Ajout d'Observer

1. Ajouter au diagramme de classes un nouvel observateur avec une Classe AppSecu qui affiche un message « Alerte » en cas d'activation
2. Ecrire en python le code de cette classe
3. Ecrire le nouveau programme principal

4. Ajout d'objets

1. Ajouter au diagramme de classes une classe Pièce associée au capteur de position, contenant un numéro et une surface.
2. Modifier la méthode detecterPresence() pour passer en paramètre la pièce pour notifier les observateurs.
3. Modifiez le code pour que plusieurs capteurs associés à différentes salles soient surveillés par les AppIot et AppSécu