TP03 – Héritage

L'objectif de ce TP est d'illustrer la notion d'**héritage,** notion importante dans le cadre de la POO. L'application de ce TP a pour contexte l'IoT et utilise le SenseHat.

Exercice 1 – Des capteurs …

L'environnement actuel, avec le développement de l'IoT, comporte énormément de composants, que ce soit des capteurs ou bien des actionneurs (ou effecteurs). Ainsi une gestion à distance et à grande échelle est envisageable puisque ces composants sont connectés. Cependant la gestion de ces composants nécessite un travail préalable de programmation pour être efficace et exploiter au mieux le fait qu'ils soient connectés.

Le travail demandé dans cet exercice aborde la structuration des données : composants et capteurs.

Nous définissons ainsi plusieurs classes :

1) La classe `Composant` : elle a pour attribut le `nom` et `hat` : une référence à l'objet SenseHat du composant et pour méthodes :
   - `__init__ (self, nom)`
   - `retournerNom(self)` : cette méthode affiche : « je suis le composant XXX » (où XXX est le nom du composant)
2) La classe `Capteur` qui hérite de la classe `Composant`. Elle définit une nouvelle méthode sans l'implémenter :
   - `afficherValeur(self)`
3) Les 3 classes `CapteurTemperature`, `CapteurPression`, `CapteurHumidite` qui héritent de la classe `Capteur`. Elles implémentent cette méthode héritée :
   - `afficherValeur(self)`

1 – Préparation avant la séance de TP

1) Faire le diagramme de classes à partir des informations données ci-dessus

2 – Réalisation en séance

1) Implémenter la structure du diagramme de classes que vous avez fait précédemment.

   Voici le code du programme qui instancie 3 capteurs :
   - Un capteur de température : `captTemperature1`
   - Un capteur de pression : `captPression1`
   - Un capteur d'humidité : `captHumidite1`

```
captTemperature1=CapteurTemperature("temp1")
captPression1=CapteurPression("pression1")
captHumidite1=CapteurHumidite("humidite1")
captTemperature1.retournerNom()
captPression1.retournerNom()
captHumidite1.retournerNom()
captTemperature1.afficherValeur()
captPression1.afficherValeur()
captHumidite1.afficherValeur()
```

a. Coder et tester votre code

b. En Déduire le diagramme de séquence

c. Que pouvez-vous dire sur la classe `Capteur` ?

2) Créer une liste de capteurs comportant 3 capteurs de température, 3 capteurs d'humidité et 2 capteurs de pression.

   Parcourir cette liste et afficher le nom et la valeur de chaque capteur

3) Ajouter la méthode `retournerNom()` de façon à ce qu'un objet de la classe `Temperature`, `Humidite`, ou `Pression` affiche le message « je suis le capteur de {température, humidité, pression} : XXX ».

   a. Tester votre solution

   b. Modifier la méthode `retournerNom`() de façon à ce que les 2 affichages se fassent (sans coder les 2 affichages dans la même fonction !)

   c. Quel est le concept mis en œuvre dans cette question ?

   d. Faire Valider les questions 2 et 3

Exercice 2 – … des Leds

Les composants englobent, en plus des capteurs, des actionneurs. Sur la carte SenseHat disponible, la matrice d'affichage composée par des leds est un exemple d'actionneur. Dans cet exercice, nous allons compléter la classe `Composant` pour intégrer la classe `Led`. Nous ne ferons pas la classe qui représente la matrice de leds.

1 – Préparation avant la séance de TP

Ajouter sur le diagramme de classes de l'exercice 1, la classe `led` qui a pour attributs :

- La position (en X et Y)
- La couleur avec les paramètres R, G, B

et pour méthodes :

- `readColor()` : qui demande à l'utilisateur les paramètres RGB de la couleur de la led
- `readPosition()` : qui demande à l'utilisateur la position de la led
- `printColor()` : qui affiche la couleur de la led

- `printPosition()` : qui affiche la position de la led
- `switchOnLed()`: qui allume sur la matrice SenseHat la led (avec la couleur et la position définies). Cette méthode utilise :

  `bool senseSetRGB565pixel(int x, int y, rgb565_pixel_t rgb565)`

Pour cela, vous avez à disposition 2 classes supplémentaires qui sont fournies ci-dessous :

- La classe `Point` qui a pour attribut les coordonnées X et Y.
- La classe `Couleur` qui a pour attribut la couleur (RGB).

```
class Point():
    def __init__(self):
        self.x=0
        self.y=0

    def lireCoordonnes(self):
        self.x=int(input("Donner l'abscisse du point"))
        self.y=int(input("Donner l'ordonnée du point"))

    def afficherCoordonnees(self):
        print(self.x, self.y)

class Couleur():
    def __init__(self):
        self.RGB=[0,0,0]

    def lireCouleur(self):
        self.RGB.clear()
        for elt in range(3):
            self.RGB.append(int(input("Entrez les couleurs R, G, B")))

    def afficherCouleur(self):
        print (self.RGB)
```

2 – Réalisation en séance

1. Après avoir recopié et renommé (avec « *save as* ») l'exercice précédent, modifier le code en rajoutant les classes `Led`, `Point` et `Couleur` .
2. Déclarer les méthodes.
3. Créer 5 leds qui sont positionnés aux angles et au centre de la matrice SenseHat. Chaque led doit avoir une couleur différente.
4. Faire valider.

Exercice 3 – … et des actionneurs

Cet exercice, a pour objectif de compléter la classe `Composant` pour intégrer en plus de la classe `Led`, d'autres actionneurs. Parmi les actionneurs on peut citer un moteur, un CNA, une électrovanne, un buzzer etc.

Ces actionneurs partagent la logique de deux fonctions : la première indique (à l'utilisateur) la **valeur de sortie** ; la seconde **désactive l'actionneur** (en cas d'urgence par ex. ou au démarrage).

1 – Préparation avant la séance de TP

1. Pour intégrer les différents types d'actionneurs dans la représentation des composants que vous avez via le diagramme de classes :
   a. Combien de classes faut-il rajouter ?
   b. Quelles sont les relations qui lient ces nouvelles classes (dont les classes `Moteur`, `CNA)` et les anciennes ?
   c. Quelle(s) modification(s) devez-vous apporter à la structure des classes ?
2. Donner le nouveau diagramme de classes modifié et complété pour intégrer, en plus de la classe `Led`, les nouvelles classes (dont `Moteur` et `CNA)`.

2 – Réalisation en séance

1. Après avoir recopié et renommé (avec « *save as* ») l'exercice précédent, modifier le code en ajoutant les classes que vous avez définies sur le diagramme de classes.
2. Déclarer les méthodes `activate(), deactivate()` et `printValue()`.
   a. `Activate()` : cette fonction affecte à la valeur de l'actionneur la valeur passée en paramètre
   b. `Deactivate()`: cette fonction affecte à la valeur de l'actionneur la valeur zéro
   c. `printValue()`: cette fonction affiche la valeur courante de l'actionneur
3. Instancier 2 moteurs, 3 CNA et 5 leds.
4. Implémenter une séquence de code qui désactive les actionneurs.
5. Faire valider.

Bonus

6. Supposons que les classes `CNA`, `Moteur` et `Led` existent mais pas la classe `actionneur`. Quelles modifications doivent être apportées au code pour désactiver tous les actionneurs (même situation que précédemment)
7. Tester et faire valider