Ajout de la classe Salle

/home/phil/Documents/LMG/STS_A_2025-2027/PYTHON_STS2/01_eval2_GESTION_PARC_INFORMATIQUE_

Table des matières

1. Situation....1
   a. Un premier problème....2
   b. Identifier les objets du problème....2
2. Création de la classe Salle....2
3. Une salle peut contenir plusieurs PC....3
4. Un objet comme attribut d'un autre objet....3
5. Modifier la classe PC....4
6. Modifier la méthode de déplacement....4
7. Représenter l'association....5
8. Comprendre les cardinalités....6
9. Généralisation : qu'est-ce qu'une association entre classes ?....6
10. Petit exercice de synthèse....6
11. À retenir....7

1. Situation

Jusqu'à présent, nous avons utilisé une classe `PC` pour représenter les ordinateurs du lycée. Chaque ordinateur possède notamment une propriété `salle.` Jusqu'à présent, cette propriété contient simplement le nom de la salle : `pc1.salle = "E03B"`

Cette solution fonctionne, mais le lycée souhaite maintenant mémoriser davantage d'informations sur les salles. Pour chaque salle, on souhaite connaître :

- son nom ;
- son bâtiment ;
- son étage ;
- une remarque éventuelle qui la caractérise.

Par exemple :

Nom	Bâtiment	Étage	Remarque
E03B	E	0	Salle informatique des BTS
F10	F	1	Salle de TP de SI
L01	L	0	Salle de réunion

Question 1:

Avec la classe `PC` actuelle, comment pourrait-on mémoriser le bâtiment, l'étage et la remarque de la salle dans laquelle se trouve un ordinateur ?

Y-a-t-il une solution `"` propre`"` ?

a. Un premier problème

On pourrait ajouter directement ces informations dans la classe `PC` :

```
class PC:

    def __init__(self, identifiant, ram_Go, stockage_Go,
                 salle, batiment, etage, remarque):
        ...
```

On aurait alors :

```
pc1 = PC("PC-001", 8, 512, "E03B", "E", 0, "Salle informatique")
```

Question 2 :

Quel problème pose cette solution si **20 ordinateurs se trouvent dans la même salle E03B** ?

Imaginez que suite à un changement d'affectation, la remarque de la salle change.

Faudrait-il modifier les 20 objets `PC` ?

b. Identifier les objets du problème

Une bonne modélisation consiste à identifier les **objets qui possèdent leur propre identité et leurs propres propriétés**. Nous avons ici deux types d'objets différents :

Un ordinateur

Un ordinateur possède :

- `identifiant`
- `date_reception`
- `ram_Go`
- `stockage_Go`
- `salle`
- `actif`
- `historique`

Une salle

Une salle possède :

- `nom`
- `bâtiment`
- `étage`
- `remarque`

Question 3 :

Justifier qu'une `Salle` devrait être représentée par une classe indépendante .

2. Création de la classe Salle

Nous allons créer une nouvelle classe :

```
class Salle:

    def __init__(self, nom, batiment, etage, remarque):
        self.nom = nom
        self.batiment = batiment
        self.etage = etage
        self.remarque = remarque
```

Question 4

Créez l'objet correspondant à la salle E03B :

```
Nom : E03B
Bâtiment : E
Étage : 0
Remarque : Salle informatique
```

On souhaite obtenir un objet :

```
salle_E03B = ...
```

3. Une salle peut contenir plusieurs PC

Nous pouvons maintenant créer plusieurs ordinateurs :

```
pc1 = PC("PC-001", 8, 512, ...)
pc2 = PC("PC-002", 16, 1000, ...)
pc3 = PC("PC-003", 8, 256, ...)
```

Les trois ordinateurs sont installés dans la même salle. Au lieu de mémoriser simplement : `"E03B"` nous allons mémoriser **l'objet `Salle` lui-même**. On souhaite donc pouvoir écrire :

```
pc1.salle = salle_E03B
```

et :

```
pc2.salle = salle_E03B
```

Question 5

Complétez la création des objets :

```
salle_E03B = Salle(...)

pc1 = PC("PC-001", 8, 512, salle_E03B)
pc2 = PC("PC-002", 16, 1000, salle_E03B)
```

4. Un objet comme attribut d'un autre objet

La propriété `salle` de `PC` ne contient désormais plus une chaîne de caractères. Elle contient un **objet de la classe `Salle`**. On peut donc écrire : `pc1.salle` pour obtenir l'objet `Salle`.

Et on peut accéder aux propriétés de cette salle :

```
pc1.salle.nom
pc1.salle.batiment
pc1.salle.etage
pc1.salle.remarque
```

Question 6

Que renvoie chacune des instructions suivantes ?

```
print(pc1.salle.nom)
print(pc1.salle.batiment)
print(pc1.salle.etage)
print(pc1.salle.remarque)
```

5. Modifier la classe PC

Il faut maintenant modifier le constructeur de `PC`.

Auparavant :

```
def __init__(self, identifiant, ram_Go, stockage_Go, salle):
```

La propriété `salle` était une chaîne de caractères.

Elle doit maintenant contenir un objet `Salle`.

Le constructeur devient :

```
def __init__(self, identifiant, ram_Go, stockage_Go, salle):
    self.identifiant = identifiant
    self.ram_Go = ram_Go
    self.stockage_Go = stockage_Go
    self.salle = salle
    ...
```

Question 7

Pourquoi n'est-il pas nécessaire de modifier la ligne :

```
self.salle = salle
```

alors que `salle` contient maintenant un objet plutôt qu'une chaîne de caractères ?

6. Modifier la méthode de déplacement

La méthode `deplacer()` permettait jusqu'à présent de déplacer un PC vers une salle en utilisant son nom :

```
pc1.deplacer("F10")
```

Désormais, nous souhaitons déplacer le PC vers **un objet `Salle`**.

Par exemple :

```
salle_F10 = Salle("F10", "F", 1, "Salle de cours")
```

On souhaite pouvoir écrire :

```
pc1.deplacer(salle_F10)
```

Question 8

Modifiez la méthode `deplacer()` pour qu'elle reçoive un objet `Salle`.

Indice :

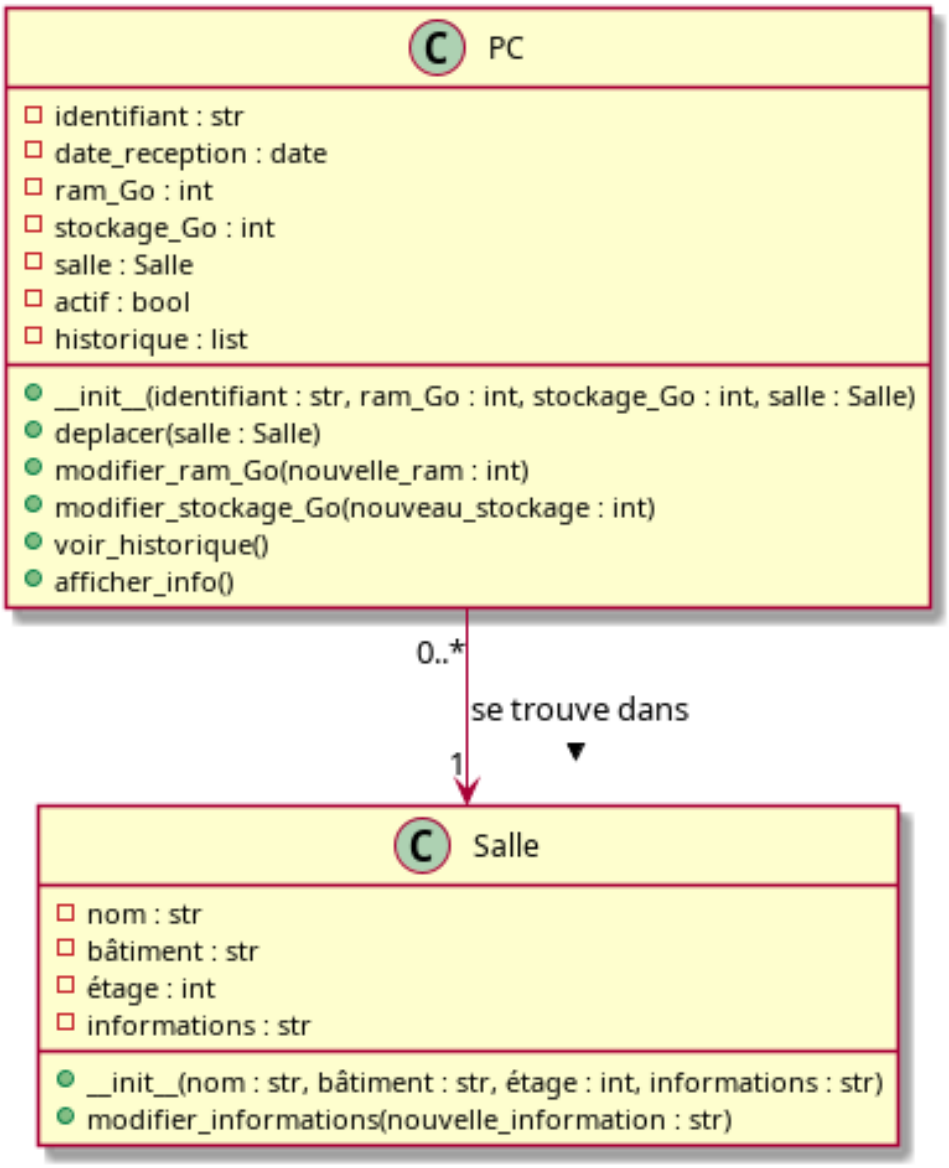
```
def deplacer(self, salle):
    ...
```

La méthode doit :

1. mémoriser l'ancienne salle ;
2. modifier `self.salle` ;
3. conserver une trace du déplacement dans l'historique ;
4. mettre à jour `actif` si nécessaire.

7. Représenter l'association

Nous avons maintenant deux classes : `PC` et `Salle`. Un PC **est associé à** une salle. Une salle peut être associée à plusieurs PC. On peut représenter cette situation dans un diagramme de classes :



8. Comprendre les cardinalités

Sur le diagramme, on trouve :

```
PC "0..*" --> "1" Salle
```

Cela indique que :

- une `Salle` peut contenir **0, 1 ou plusieurs PC** ;
- un `PC` est associé à **une Salle**.

Question 9

Dans le contexte du lycée, que signifie : `"0..*"` et que signifie : `"1"`

9. Généralisation : qu'est-ce qu'une association entre classes ?

Nous venons de créer une **association entre deux classes**.

Une association signifie qu'un objet d'une classe est **lié à un ou plusieurs objets d'une autre classe**.

Dans notre exemple :

```
PC ————————————> Salle
```

Un objet `PC` possède une référence vers un objet `Salle`.

Mais ce principe est général.

Dans un autre programme, on pourrait avoir par exemple :

```
Étudiant ————————————> Classe
```

ou :

```
Livre ————————————> Auteur
```

ou :

```
Commande ————————————> Client
```

ou encore :

```
Voiture ————————————> Conducteur
```

Dans tous ces cas, une classe utilise une autre classe pour représenter une relation entre les objets.

10. Petit exercice de synthèse

On souhaite maintenant gérer les salles suivantes :

```
E03B – Bâtiment E – étage 0 – Salle informatique
F10  – Bâtiment F – étage 1 – Salle de cours
B02  – Bâtiment B – étage 0 – Salle de réunion
```

Travail demandé

1. Créer les trois objets `Salle`.
2. Créer trois objets `PC`.
3. Placer deux PC dans la salle E03B.
4. Placer un PC dans la salle F10.
5. Afficher pour chaque PC :
    - son identifiant ;
    - le nom de sa salle ;
    - le bâtiment ;
    - l'étage.

Le résultat attendu pourrait avoir cette forme :

```
PC-001
Salle : E03B
Bâtiment : E
Étage : 0
```

```
PC-002
Salle : E03B
Bâtiment : E
Étage : 0

PC-003
Salle : F10
Bâtiment : F
Étage : 1
```

11. À retenir

Une classe permet de représenter un **type d'objet** et ses propriétés. Lorsqu'un objet possède une propriété qui correspond à **un autre objet**, on peut créer une association entre les deux classes.
Dans notre exemple :

```
pc1.salle = salle_E03B
```

signifie que :

- `pc1` est un objet de la classe `PC` ;
- `salle_E03B` est un objet de la classe `Salle` ;
- la propriété `salle` de `pc1` contient une référence vers `salle_E03B`.
-

On obtient donc :

```
        possède ▶
PC ──────────────────> Salle
```

C'est une **association entre deux classes**.