PROGRAMMATION D'INTERFACES

Licence informatique & vidéoludisme

Cours préparé par:
Oumaima EL JOUBARI
Hanane ZERDOUM

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Programmation orientée objet

01

Introduction

Introduction à la POO
Concepts de base de la POO

02

Classes et instances

Définition des classes
Constructeur de classe

03

Les méthodes

Définition et appel des méthodes

04

Héritage et polymorphisme

Héritage
Classes et sous-classes
Polymorphisme et surcharge

01

Introduction

1. Introduction à la POO
2. Concepts de base de la POO

I. Introduction

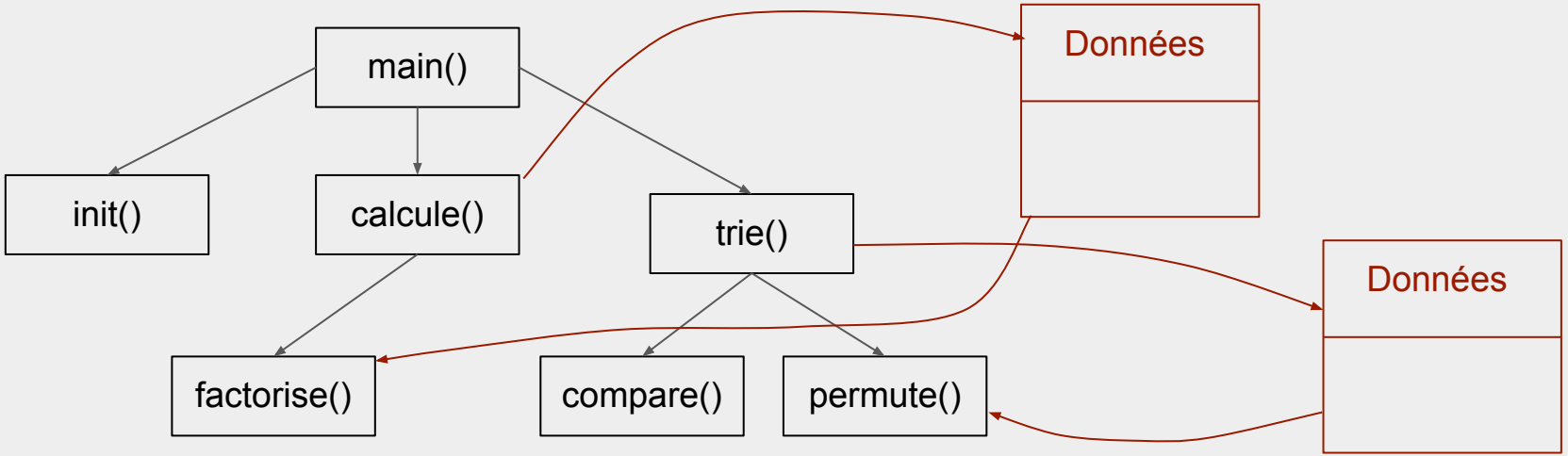
1. Introduction à la POO

La programmation procédurale (PP)

La PP repose sur l'équation suivante:

Programme = Structures de données + Algorithmes

➔ Consiste à décomposer le programme en fonctions (modules) simples.



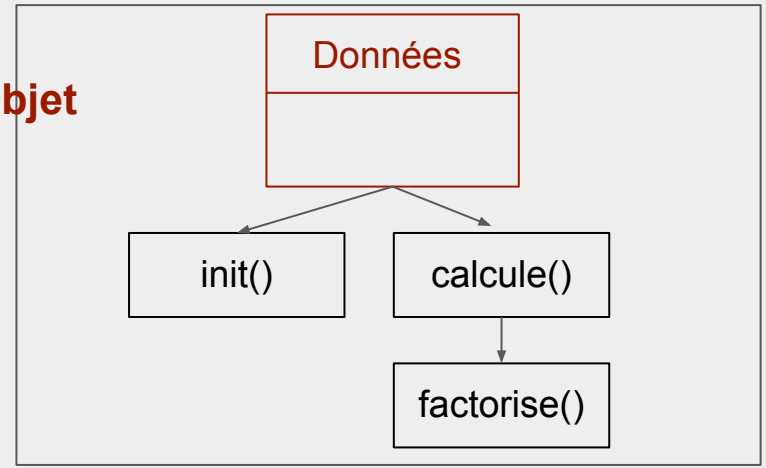
I. Introduction

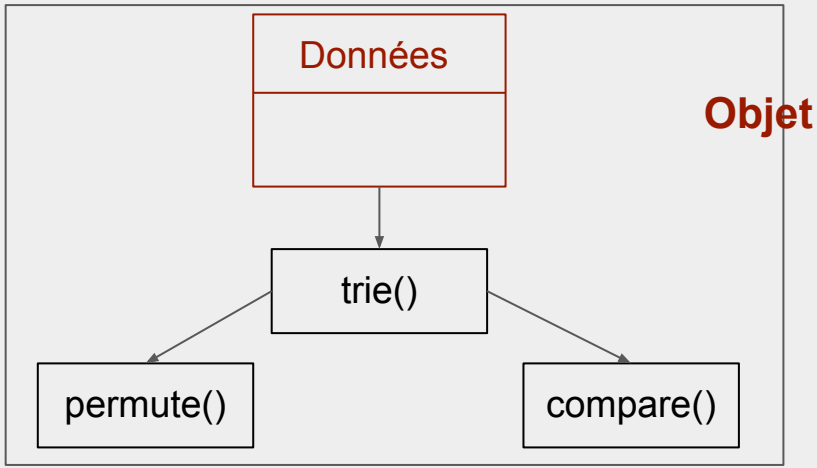
1. Introduction à la POO

La programmation orientée objet (POO)

La POO est basée sur les données:

➔ Le programme détermine les données à traiter et les fonctions qui permettent de les manipuler.





I. Introduction

1. Introduction à la POO

La programmation orientée objet (POO)

Objet= Données + Méthodes

➔ Un objet est une association de données et de fonctions (méthodes) qui agissent sur ces données.

➔ La POO est donc une programmation dans laquelle un programme est organisé comme un ensemble d'objets coopérant ensemble.

I. Introduction

2. Concepts de base de la POO

➔ Un objet a:

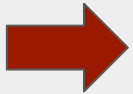
- Ses propres données (*attributs*) qui peuvent être des données simples (entier, chaîne de caractère, …) ou d'autres objets.
- Ses propres fonctions membres (*méthodes*) qui représente son comportement.
  - Ce sont les traitements qu'on peut appliquer aux données.
- Une *identité* qui permet de l'identifier parmi les autres objets.

I. Introduction

2. Concepts de base de la POO

➔ Un objet a:

- ◆ Ses propres données (*attributs*) qui peuvent être des données simples (entier, chaîne de caractère, …) ou d'autres objets.
- ◆ Ses propres fonctions membres (*méthodes*) qui représente son comportement.
  - Ce sont les traitements qu'on peut appliquer aux données.
- ◆ Une *identité* qui permet de l'identifier parmi les autres objets.



Voiture1

Marque: Lamborghini
Modèle: Aventador
Couleur: Noir
Etat: En Marche

Démarrer()
Arrêter()
Peindre(NouvColor)
GetEtat()
…

I. Introduction

2. Concepts de base de la POO

- ➔ Objets prédéfinis par Python:
  - ◆ Entiers, listes, booléens, chaînes de caractères...
- ➔ Pour créer un nouveau type d'objets, il faut définir à quoi il ressemble = *définir une classe*.
- ➔ Les *classes* servent de « moules » pour la création des objets.
- ➔ Les objets qui ont les mêmes états et les mêmes comportements sont regroupés : classe.

02

Classes et instances

1. Définition d'une classe
2. Constructeur de classe

II. Classes et instances

1. Définition d'une classe:

Classe

Anime
Titre (chaîne de caractères) **Genre** (chaîne de caractères) **Date de 1ère diffusion** (entier) **Auteur** (chaîne de caractères)

II. Classes et instances

1. Définition d'une classe:

Classe

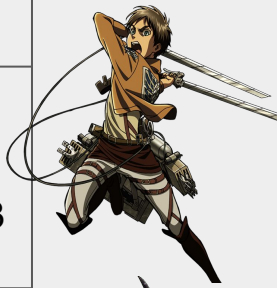
Anime

Titre (chaîne de caractères)
Genre (chaîne de caractères)
Date de 1ère diffusion (entier)
Auteur (chaîne de caractères)

Instances

AOT

Titre : "Attack on titan"
Genre : "Shonen"
Date de la 1ère diffusion : 2013
Auteur : "Hajime Isayama"

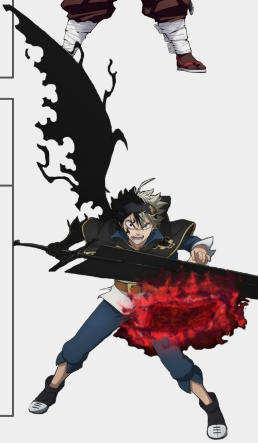


DS

Titre : "Demon slayer"
Genre : "Shonen"
Date de la 1ère diffusion : 2019
Auteur : "Koyoharu Gotoge"

BC

Titre : "Black clover"
Genre : "Shonen"
Date de la 1ère diffusion : 2017
Auteur : "Yuki Tabata"



II. Classes et instances

1. Définition d'une classe:

➔ Créer une classe:

```python
#Définition de la classe Voiture
class Voiture:
    #Les attributs de la classe
    marque = "Lamborghini"
    modele = "Aventador"
    couleur = "Noir"

#Création de deux instances de la classe Voiture
voiture_1 = Voiture()
voiture_2 = Voiture()

#Afficher la marque des deux voitures crées
print(voiture_1.marque)
print(voiture_2.marque)
```

Exécution

```
(base) Oumaimas-MacBook-Air:desktop oumaima$ python Voiture.py
Lamborghini
Lamborghini
```

II. Classes et instances

2. Constructeur de la classe:

➔ Créer un constructeur de classe:

```python
#Définition de la classe Voiture
class Voiture:
    #création du constructeur
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

#Création de deux instances de la classe Voiture
voiture_1 = Voiture("Lamborghini" , "Noir")
voiture_2 = Voiture("BMW", "Rouge")

#Afficher la marque des deux voitures crées
print(voiture_1.marque)
print(voiture_1.couleur)
print(voiture_2.marque)
print(voiture_2.couleur)
```

Exécution

```
(base) Oumaimas-MacBook-Air:partie2 oumaima$ python Voiture2.py
Lamborghini
Noir
BMW
Rouge
```

03

Les méthodes

Définition et appel des méthodes

III. Les méthodes

- Définition et appel des méthodes

```python
#Definition de la classe Voiture
class Voiture:

    #creation du constructeur
    def __init__(self, marque, couleur):
        self.marque = marque
        self.couleur = couleur

    #Definition des methodes

    def peindre(self, NouvCouleur):
        self.couleur = NouvCouleur

#Creation de deux instances de la classe Voiture
voiture_1 = Voiture("Lamborghini" , "Noir")

#Afficher la marque des deux voitures crees
print(voiture_1.couleur)
voiture_1.peindre("Rouge")
print(voiture_1.couleur)
```

Exécution

```
(base) Oumaimas-MacBook-Air:partie2 oumaima$ python Voiture4.py
Noir
Rouge
```

04

Héritage et polymorphisme

1. Héritage
2. Classes et sous-classes
3. Polymorphisme et surcharge

IV. Polymorphisme et héritage

1. Définition et intérêts

➔ Définition:

Technique offerte par les langages de programmation pour construire une classe à partir d'une autre en partageant ses attributs et méthodes.

➔ Intérêts:

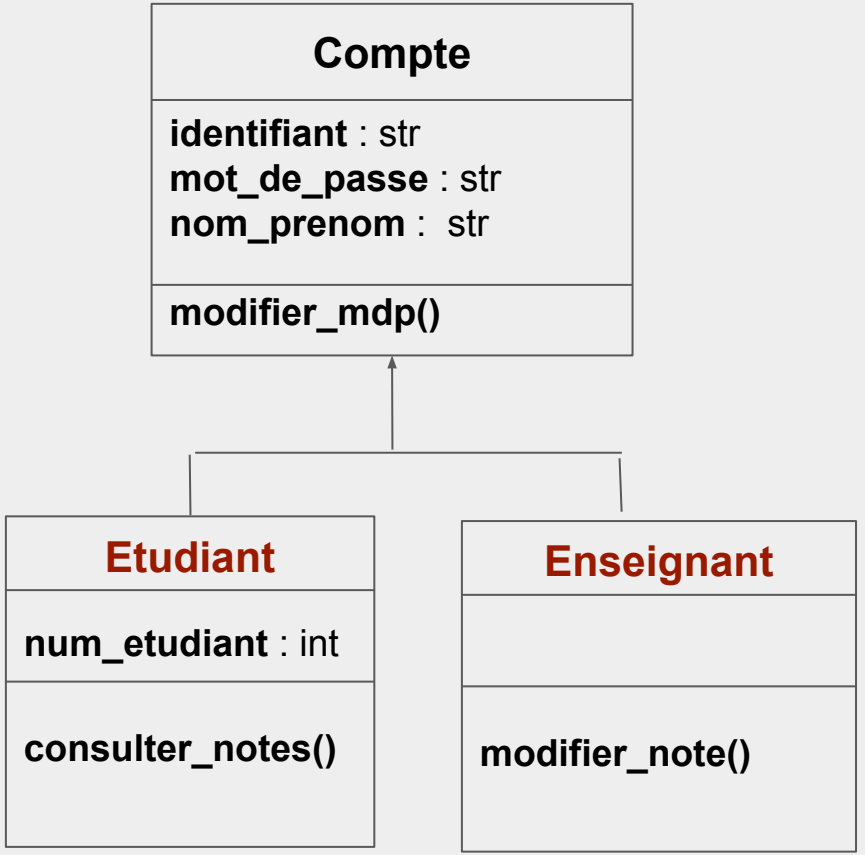
- **Spécialisation:** une nouvelle classe hérite les attributs/méthodes de la classe mère mais on peut lui ajouter de nouveaux attributs/méthodes.
- **Redéfinition:** une nouvelle classe peut redéfinir les attributs/méthodes d'une classe de manière à en changer le sens ou le comportement pour le cas particulier défini par la nouvelle classe.
- **Réutilisation:** évite d'avoir à répéter du code existant de façon inutile.

IV. Polymorphisme et héritage

2. Classes et sous-classes

➔ <u>Définitions:</u>

- Les classes *Etudiant* et *Enseignant* héritent de la classe *Compte*.
- La classe *Compte* est appelée la classe mère.
- Les classes *Etudiant* et *Enseignant* sont appelées les classes filles.
- La classe Compte est la super-classe des classes *Etudiant* et *Enseignant*.
- *Etudiant* et *Enseignant* sont les sous-classes de la classe *Compte.*

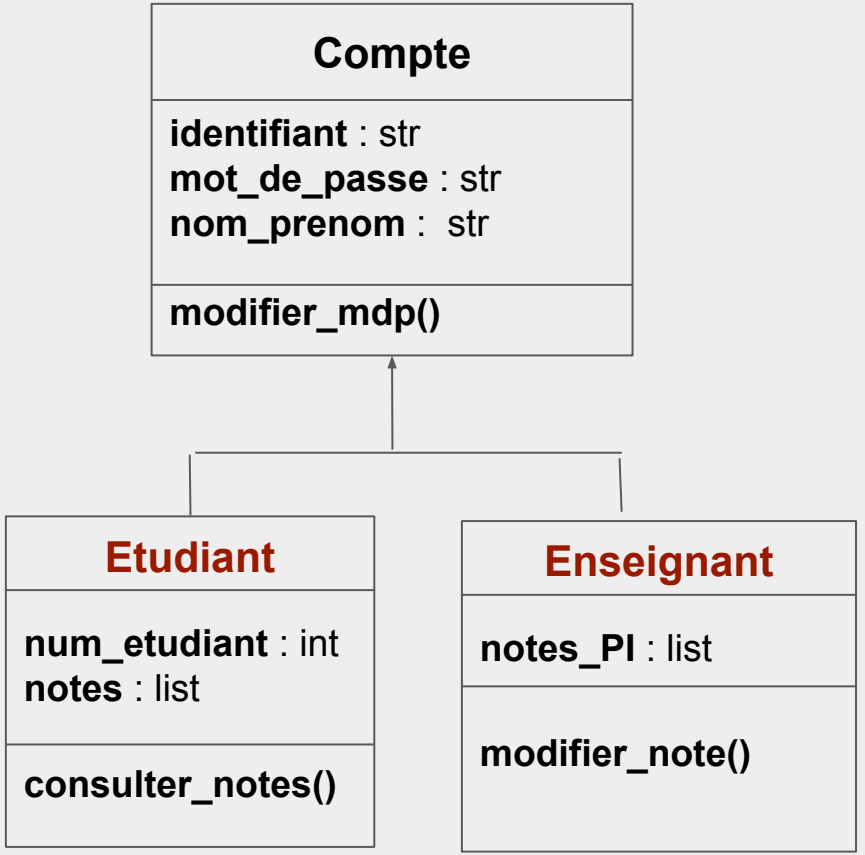


IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Définition de la classe Compte
class Compte:
    def __init__(self, id, mdp, nom_prenom):
        self.id = id
        self.mdp = mdp
        self.nom_prenom = nom_prenom

    def modifier_mdp(self, NouvMdp):
        self.mdp = NouvMdp
```

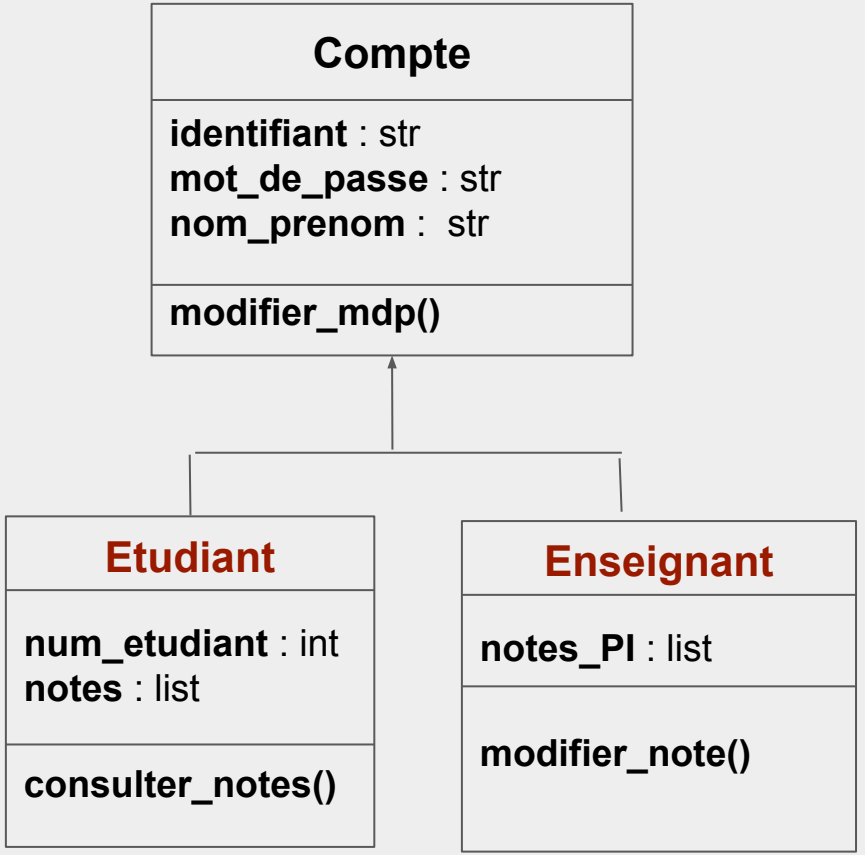


IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Definition de la classe Etudiant
class Etudiant(Compte):
    def __init__(self, id, mdp, nom_prenom, num_etudiant, notes):
        super().__init__(id, mdp, nom_prenom)
        self.num_etudiant = num_etudiant
        self.notes = notes

    def consulter_notes(self):
        print(self.notes)
```



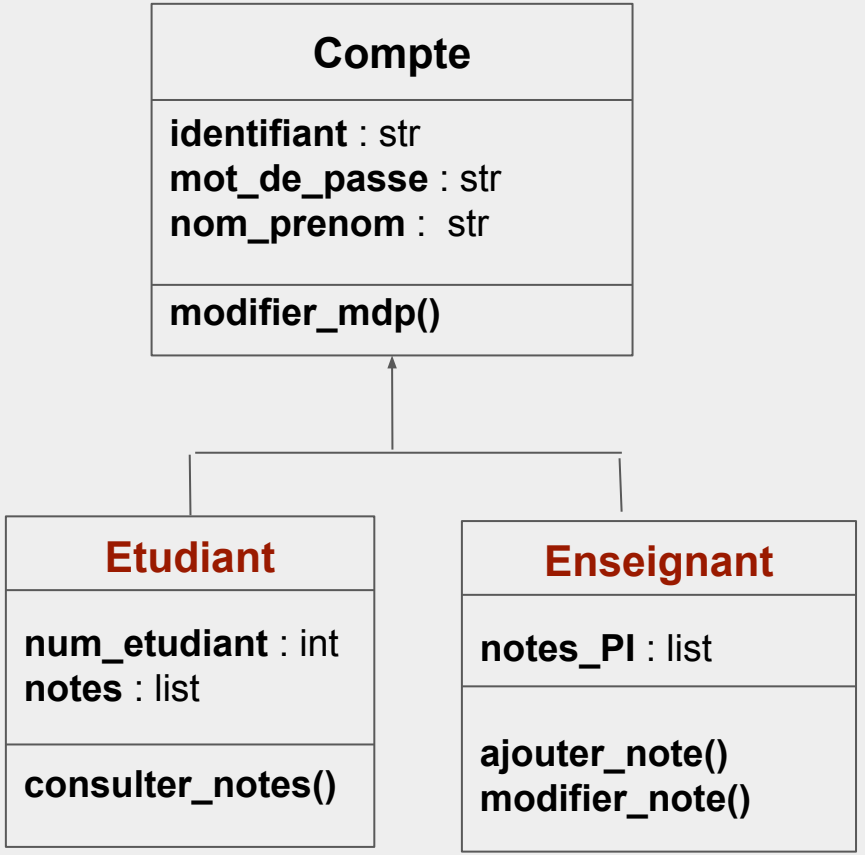
IV. Polymorphisme et héritage

2. Classes et sous-classes

```python
#Definition de la classe Enseignant
class Enseignant(Compte):
    def __init__(self, id, mdp, nom_prenom, notes_PI):
        super().__init__(id, mdp, nom_prenom)
        self.notes_PI = []

    def ajouter_note(self, note):
        self.notes.append(note)

    def modifier_note(self, NouvNote, etudiant):
        self.notes_PI[etudiant] = NouvNote
```



IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

➔ Définition du polymorphisme: C'est la capacité d'une méthode à se comporter différemment en fonction de l'objet qui lui est passé.

➔ Exemple: La méthode *sorted()*

Cette méthode trie par ordre ASCII les chaînes de caractères et par ordre croissant les listes d'entiers:

```
liste_triee = sorted([13,4,78,12,98])
```

```
liste_triee
```

```
[4, 12, 13, 78, 98]
```

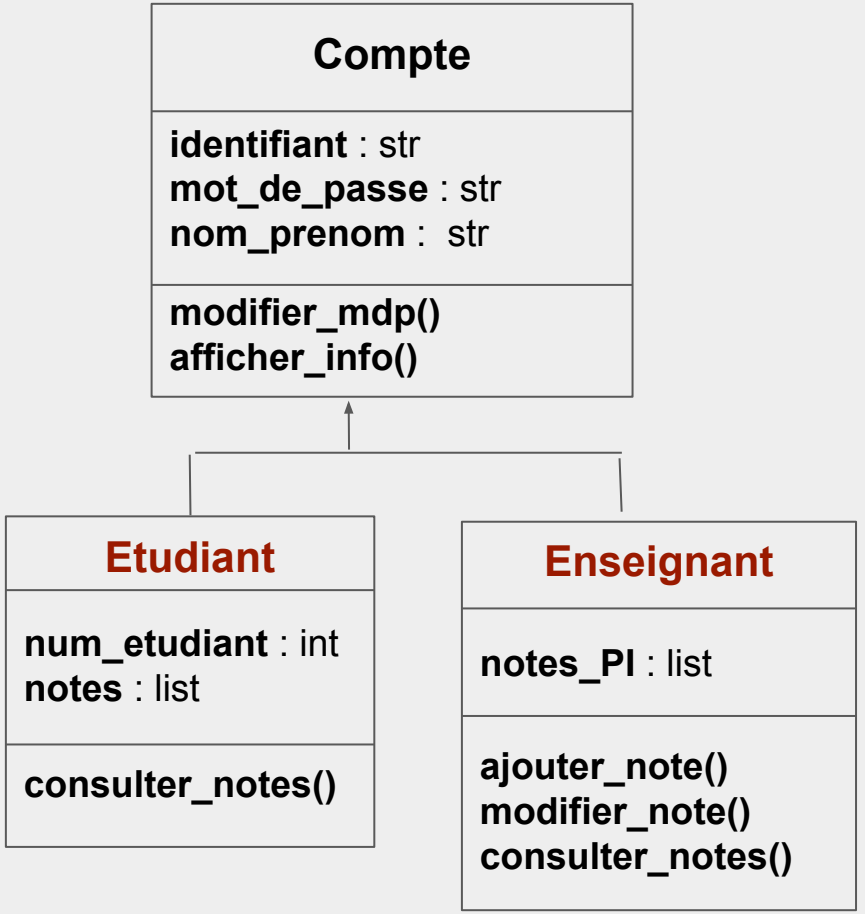
```
liste_triee = sorted("bonjour")
```

```
liste_triee
```

```
['b', 'j', 'n', 'o', 'o', 'r', 'u']
```

IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

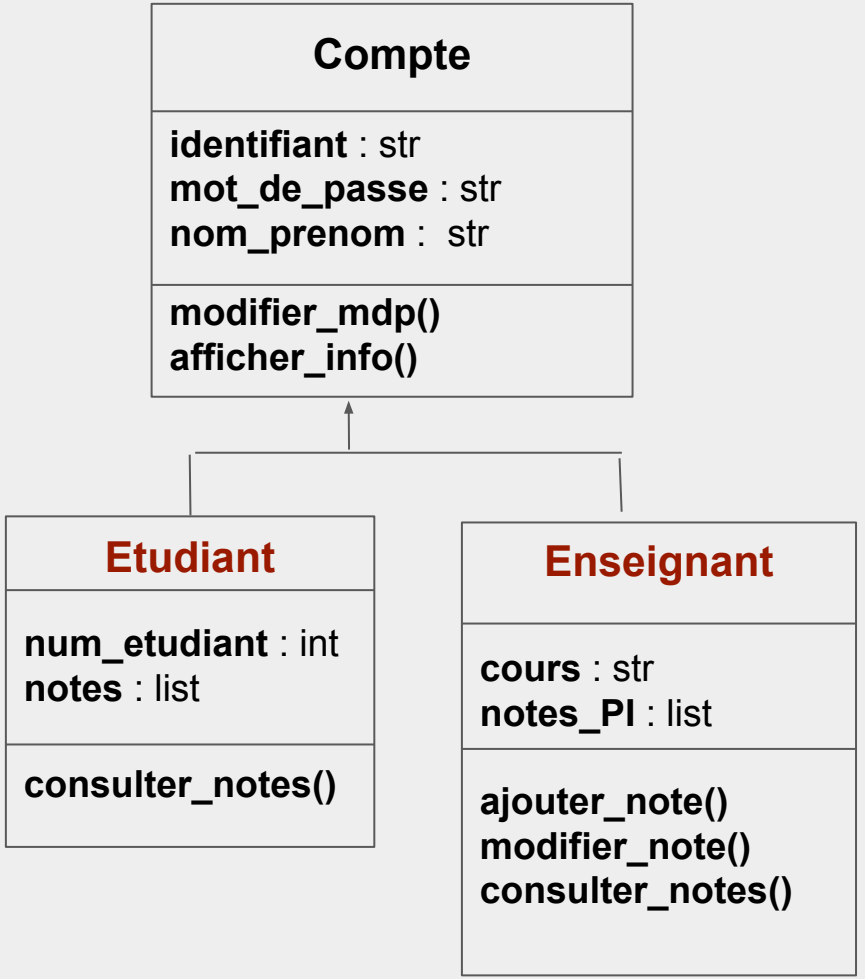


IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

Définition de la surcharge: possibilité de définir des méthodes possédant le même nom mais des arguments différents.

Redéfinition (overriding): lorsque la sous-classe définit une méthode dont le nom et les paramètres sont identiques.



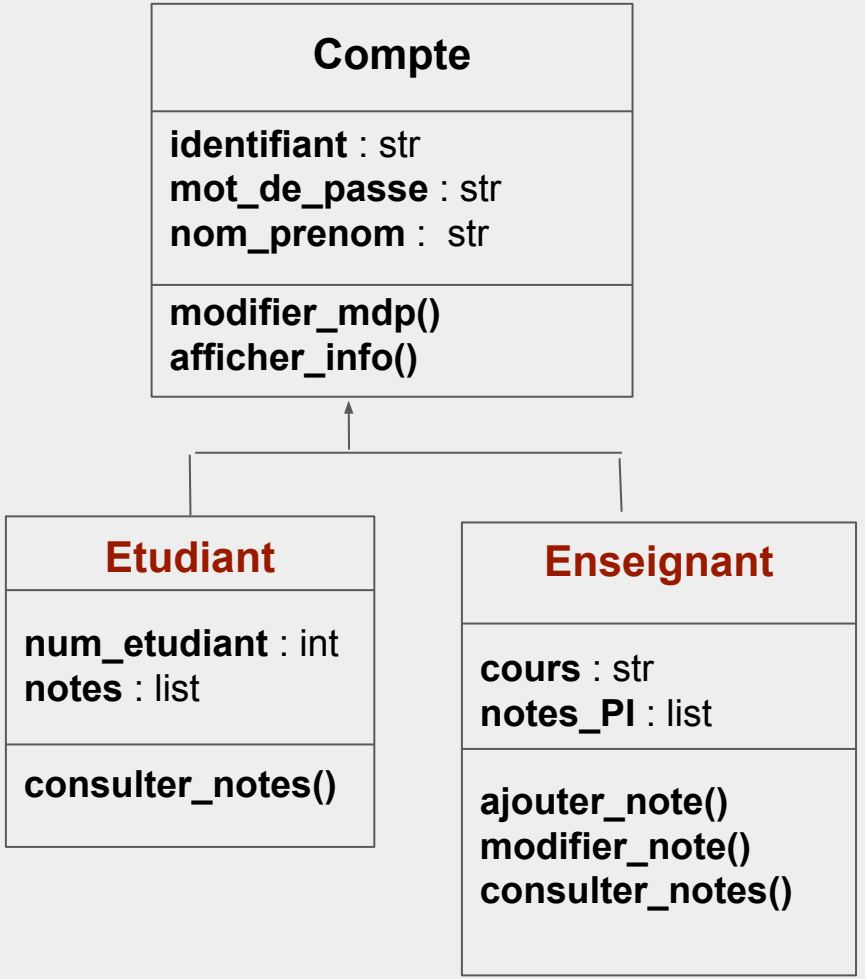
IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
class Compte:
    def __init__(self, id, mdp, nom_prenom):
        self.id = id
        self.mdp = mdp
        self.nom_prenom = nom_prenom

    def modifier_mdp(self, NouvMdp):
        self.mdp = NouvMdp

    def afficher_info(self):
        print("Identifiant: ", self.id)
        print("Nom et prenom: ", self.nom_prenom)
```



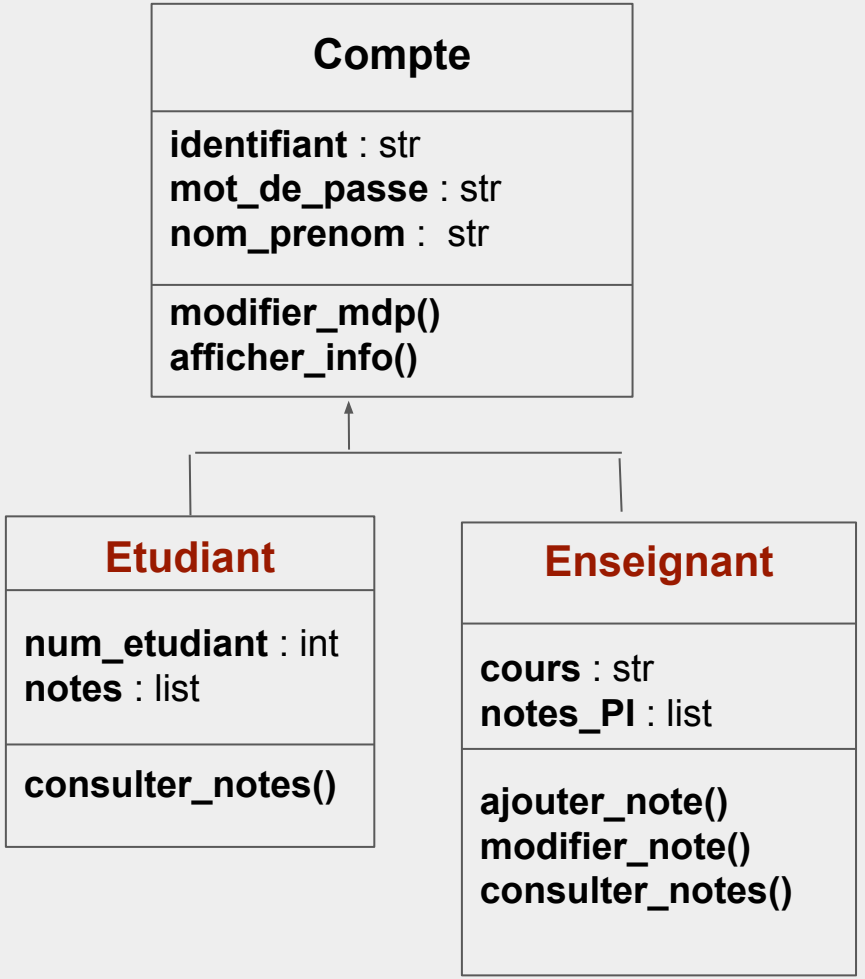
IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
#Definition de la classe Etudiant
class Etudiant(Compte):
    def __init__(self, id, mdp, nom_prenom, num_etudiant, notes):
        super().__init__(id, mdp, nom_prenom)
        self.num_etudiant = num_etudiant
        self.notes = notes

    def consulter_notes(self):
        print(self.notes)

    def afficher_info(self):
        super().afficher_info()
        print("Numero-etudiant: ", self.num_etudiant)
```



IV. Polymorphisme et héritage

3. Polymorphisme et surcharge

```python
#Definition de la classe Enseignant
class Enseignant(Compte):
    def __init__(self, id, mdp, nom_prenom, cours, notes_PI):
        super().__init__(id, mdp, nom_prenom)
        self.cours = cours
        self.notes_PI = []

    def ajouter_note(self, note):
        self.notes.append(note)

    def modifier_note(self, NouvNote, etudiant):
        self.notes_PI[etudiant] = NouvNote

    def consulter_notes(self):
        print(self.notes_PI)

    def afficher_info(self):
        super().afficher_info()
        print("Cours enseigné: ", self.cours)
```