Chapitre 3:
Classe abstraite

Mr. CHAOULID

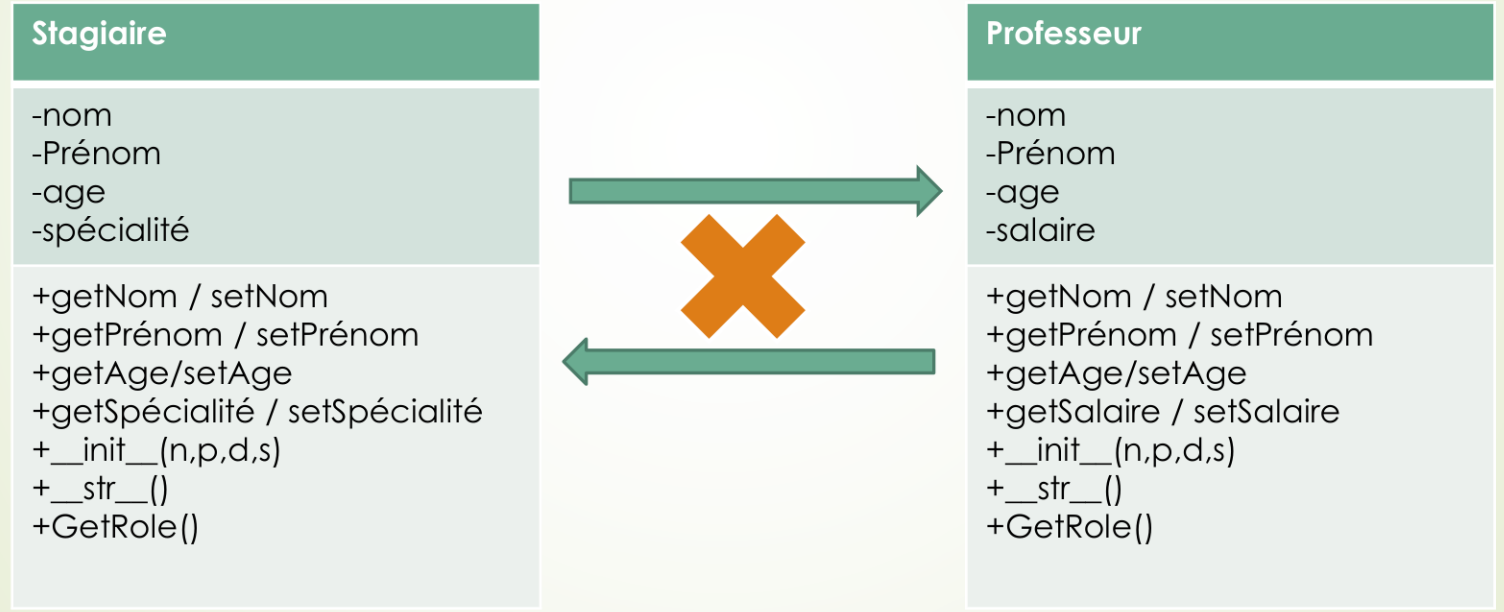
1. Classe abstraite

- Dans certains cas, lors de l'élaboration d'une hiérarchie de classe ayant un comportement commun, il peut être intéressant de mettre en facteur l'état et comportement commun à plusieurs classes, c'est-à-dire créer une **classe générique.**

- Lorsque cette classe ne correspond pas à une réalité au niveau conceptuel, elle sera dite **abstraite** et ne pourra faire l'objet d'une instanciation.

1. Classe abstraite

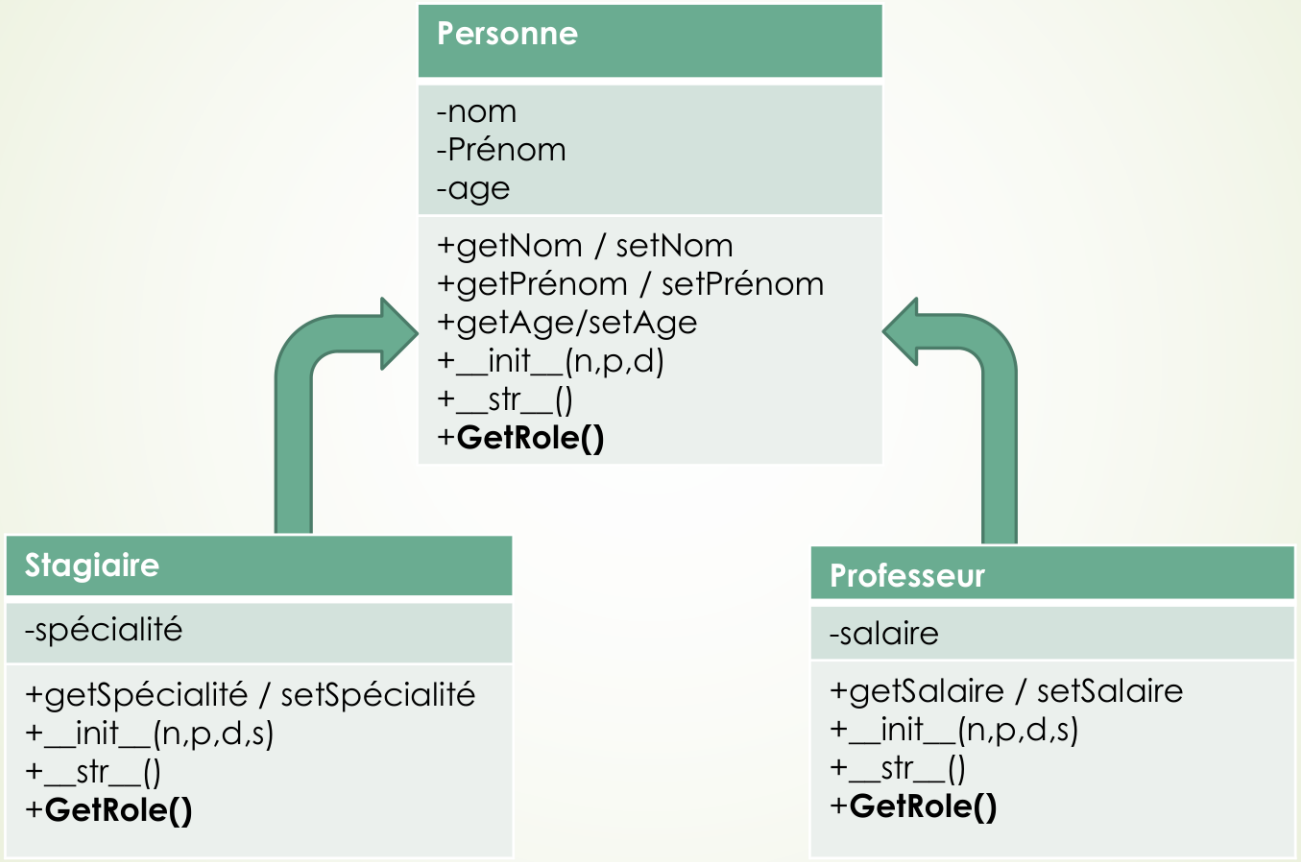
- Exemple d'application: sans classe abstraite



- Aucune relation d'héritage entre les deux classes: pas de réutilisation du code.

1. Classe abstraite

- **Exemple d'application: avec classe abstraite**



- Relation d'héritage entre les classes: réutilisation du code.

1. Classe abstraite

- En Python, Les classes abstraites sont définies en utilisant le module **abc (Abstract Base Classes).** Elle peut contenir deux types de méthodes :

  - **des méthodes non abstraites**, complètement définies, qui représentent soit des comportements par défaut pour l'héritage, soit un comportement commun hérité.
  - **des méthodes abstraites** : c'est-à-dire des méthodes sans implémentation, elles jouent le rôle d'un outil de spécification, en forçant toute sous-classe instanciable à implémenter le comportement correspondant.

1. Classe abstraite

- Exemple : La classe Abstraite Personne

```
from abc import ABC,abstractmethod
class Personne(ABC):

    def __init__(self,n,p,a):
        self.__nom=n
        self.__prenom=p
        self.__age=a

    def get_nom(self):
        return self.__nom
    def set_nom(self,n):
        self.__nom=n
    ……..
    def __str__(self):
        return f"Nom:{self.__nom},Prenom:{self.__prenom},Age:{self.__age}"

    @abstractmethod
    def getRole():
        pass
```

Méthodes non abstraites

Méthode abstraite

1. Classe abstraite

- Exemple : La classe stagiaire (hérite de la classe abstraite Personne)

```
class Stagiaire(Personne):
    def __init__(self, n, p, a,s):
        super().__init__(n, p, a)
        self.__specialite=s


    def get_specialite(self):
        return self.__specialite
    def set_specialite(self,s):
        self.__specialite=s

    def __str__(self):
        return super().__str__()+ f"specialité :{self.__specialite}"

    def getRole(self):
        print("Mon profil est Stagiaire")
```

Définition de la méthode abstraite

1. Classe abstraite

- Exemple : La classe Professeur (hérite de la classe abstraite Personne)

```
class professeur(Personne):
    def __init__(self, n, p, a,s):
        super().__init__(n, p, a)
        self.__salaire=s


    def get_salaire(self):
        return self.__salaire
    def set_salaire(self,s):
        self.__salaire=s

    def __str__(self):
        return super().__str__()+ f", salaire :{self.__salaire}"

    def getRole(self):
        print("Mon métier est professeur")
```

Définition de la méthode abstraite

1. Classe abstraite

- **Exemple: Le Programme.**

```
s=Stagiaire("Mohamed","Ali",20,"DEV")
print(s)
print(s.get_specialite())
s.getRole()

p=professeur("Mohamed","Ali",20,5000)
print(p)
print(p.get_salaire())
p.getRole()
```

- **Remarque** :Essayer d'instancier la classe abstraite « Personne » générera une **erreur.**

1. Classe abstraite

Les classes **abstraites** présentent les particularités suivantes :

- Une classe **abstraite** ne peut pas être instancié.
- Les méthodes **abstraites** n'ont pas de définition (**pass** en Python).
- Une classe peut être déclarée **abstraite**, même si elle ne comporte pas de méthodes **abstraite.**
- Pour pouvoir être instanciée, une classe fille d'une classe **abstraite** doit redéfinir toutes les méthodes **abstraite** de la classe mère.
- Si une des méthodes n'est pas redéfinie de façon concrète, la sous-classe est elle-même **abstraite** et doit être déclarée explicitement comme telle.

TP4

Mr. CHAOULID