TP4 : Héritage et Polymorphisme en Python

Objectif du TP

- Comprendre l'héritage en Python
- Utiliser le polymorphisme (surcharge et redéfinition des méthodes)
- Développer un programme orienté objet structuré et modulaire

Énoncé du TP : Une école souhaite gérer **différents types de personnes** : étudiants et enseignants. Chaque personne a des informations de base, mais les étudiants et enseignants ont des attributs et comportements spécifiques.

1. Créer une classe Personne avec :

Attributs privés :

- nom (string)
- prenom (string)
- age (int)

Méthodes publiques :

1. **__init__(self, nom, prenom, age)** : constructeur pour initialiser les attributs
2. **afficher(self)** : affiche le nom, prénom et âge

2. Créer une classe **Etudiant** qui hérite de Personne.

Attributs supplémentaires :

- numero : numéro d'inscription
- notes : liste des notes

Méthodes :

1. **saisir_notes(self)** : saisir les notes de l'étudiant
2. **calculer_moyenne(self)** : retourne la moyenne des notes
3. **afficher(self)** : redéfinir la méthode pour afficher les informations de la personne + numéro + moyenne

3. Créer une classe **Enseignant** qui hérite de **Personne.**

Attributs supplémentaires :

- **matiere** : matière enseignée
- **salaire** : salaire mensuel

Méthodes :

1. **afficher(self)** : redéfinir pour afficher les informations de la personne + matière + salaire

4. Créer une fonction **afficher_personne(p)** qui prend un objet de type Personne ou ses dérivés et appelle sa méthode **afficher().**

- Cette fonction doit fonctionner quel que soit le type de la personne (Etudiant ou Enseignant).

5. Programme principal

1. Créer au moins :
   - 2 étudiants
   - 2 enseignants
2. Saisir les notes des étudiants
3. Afficher les informations de toutes les personnes en utilisant la fonction **afficher_personne**

La solution

```python
# Classe de base : Personne
class Personne:
    def __init__(self, nom, prenom, age):
        self.__nom = nom
        self.__prenom = prenom
        self.__age = age

    def afficher(self):
        print(f"Nom : {self.__nom}, Prénom : {self.__prenom}, Âge : {self.__age}")


# Classe dérivée : Etudiant
class Etudiant(Personne):
    def __init__(self, nom, prenom, age, numero):
        super().__init__(nom, prenom, age)
        self.__numero = numero
        self.__notes = []

    def saisir_notes(self):
        n = int(input(f"Combien de notes pour l'étudiant {self.__numero} ? "))
        self.__notes = []
        for i in range(n):
            note = float(input(f"Entrer la note {i+1} : "))
            self.__notes.append(note)

    def calculer_moyenne(self):
        if len(self.__notes) == 0:
            return 0
        return sum(self.__notes) / len(self.__notes)

    # Redéfinition de la méthode afficher
    def afficher(self):
        super().afficher()
        print(f"Numéro d'inscription : {self.__numero}, Moyenne : {self.calculer_moyenne():.2f}")


# Classe dérivée : Enseignant
class Enseignant(Personne):
    def __init__(self, nom, prenom, age, matiere, salaire):
        super().__init__(nom, prenom, age)
        self.__matiere = matiere
        self.__salaire = salaire

    # Redéfinition de la méthode afficher
    def afficher(self):
        super().afficher()
        print(f"Matière : {self.__matiere}, Salaire : {self.__salaire} DZD")


# Fonction polymorphe
def afficher_personne(p):
    p.afficher()
    print("-" * 40)
# Programme principal
# Création des étudiants
etu1 = Etudiant("Dupont", "Ahmed", 20, 101)
etu2 = Etudiant("Martin", "Fatma", 22, 102)

# Saisie des notes
etu1.saisir_notes()
etu2.saisir_notes()

# Création des enseignants
ens1 = Enseignant("youcef", "naas", 35, "Mathématiques", 50000)
ens2 = Enseignant("yacine", "ahmed", 40, "Physique", 60000)

# Liste de toutes les personnes
personnes = [etu1, etu2, ens1, ens2]

# Affichage des informations avec polymorphisme
for p in personnes:
    afficher_personne(p)
```