كلية العلوم و التقنيات بطنجة
Faculté des Sciences et Techniques de Tanger

جامعة عبد المالك السعدي
Université Abdelmalek Essaâdi

Compte Rendu de TP

TP 7 : Programmation Orientée Objet (POO)

Module : Programmation Python

Pr. : Ouafae EL BOUHADI

Étudiant : Boulahrouf Hamza

Cycle d'Ingénieur en Géoinformation

Année universitaire 2025/2026

Introduction

Ce compte rendu présente les travaux réalisés dans le cadre du **TP 7 de Programmation Python**, consacré à la **Programmation Orientée Objet (POO)**. L'objectif principal était de manipuler les notions d'héritage, de polymorphisme et d'encapsulation à travers deux exercices portant sur :

— La gestion d'un système domotique intelligent ;
— La gestion d'une bibliothèque moderne.

Chaque exercice met en pratique une modélisation orientée objet complète.

Exercice 1 : Système domotique intelligent

Objectif

Concevoir une application Python permettant de gérer :
— Des lumières,
— Des climatiseurs,
— Des caméras.

L'exercice utilise l'héritage, la redéfinition de méthodes et le polymorphisme.

Code principal

```python
class Appareil:
    def __init__(self,nom):
        self.nom=nom
        self.etat=False
    def allumer(self):
        self.etat=True
    def eteindre(self):
        self.etat = False
    def etat(self):
        print(f"{self.nom} : État = {self.etat}")


class Lumiere(Appareil):
    def __init__(self,nom,intensite):
        super().__init__(nom)
        self.intensite=0
    def etat(self):
        print(f"{self.nom} : État = {self.etat} : intensite = {self.intensite}")

class Climatiseur(Appareil):
    def __init__(self,nom,temperature,mode):
        super().__init__(nom)
        self.temperature=temperature
        self.mode=mode
    def etat(self):
        print(f"{self.nom} : État = {self.etat} : temperature = {self.temperature} : mode : {self.mode}" )

class Camera(Appareil):
    def __init__(self,nom,resolution,enregistrement=False):
        super().__init__(nom)
        self.resolution=0
        self.enregistrement=enregistrement
    def etat(self):
        print(f"{self.nom} : État = {self.etat} : resolution = {self.resolution} : enregistrement : {self.enregistrement}" )
```

Code de test

```python
# ---------- TEST DE L'EXERCICE 1 : SMART HOME ----------

# Appareil, Light, AC, Camera

# Création des appareils
l1 = Lumiere("Lumière Salon", intensite=70)
l2 = Lumiere("Lumière Cuisine", intensite=30)

ac1 = Climatiseur("Climatiseur Chambre", temperature=24, mode="Froid")
ac2 = Climatiseur("Climatiseur Salon", temperature=28, mode="Chaud")

cam1 = Camera("Caméra Entrée", resolution="1080p", enregistrement=False)
cam2 = Camera("Caméra Jardin", resolution="4K", enregistrement=False)

# Tester allumer / éteindre
l1.allumer()
ac1.allumer()
cam1.eteindre()

# Afficher l'état des appareils (polymorphisme)
print("\n--- ÉTATS DES APPAREILS ---")
print(l1.etat)
print(l2.etat)
print(ac1.etat)
print(ac2.etat)
print(cam1.etat)
print(cam2.etat)
```

Résultat obtenu (exécution)

```
--- ÉTATS DES APPAREILS ---
True
False
True
False
False
False
```

Analyse

— Le polymorphisme fonctionne grâce à la méthode `etat()` redéfinie.
— L'héritage centralise les attributs communs dans `Appareil`.
— Chaque sous-classe enrichit le comportement avec ses propres propriété

Exercice 2 : Gestion d'une bibliothèque moderne

Objectif

Créer une modélisation orientée objet permettant de gérer :
— Livres,
— Magazines,
— Bandes dessinées.
Avec emprunt, retour et affichage.

Code principal

```python
class Document:
    def __init__(self,Titre,Auteur,Annee,ISBN):
        self.Titre=Titre
        self.Auteur=Auteur
        self.Annee=Annee
        self.ISBN=ISBN
        self.disponible = True

    def afficher(self):
        print(f"Titre : {self.Titre}")
        print(f"Auteur : {self.Auteur}")
        print(f"Année : {self.Annee}")
        print(f"ISBN : {self.ISBN}")

    def emprunter(self):
        if self.disponible == False:
            return print(f"Le document '{self.Titre}' est déjà emprunté.")
        else:
            self.disponible = False
            return print(f"Le document '{self.Titre}' a été bien emprunté.")

    def retourner(self):
        if self.disponible == False:
            self.disponible = True
            return print(f"Le document '{self.Titre}' a été bien retourné.")
        else:
            return print(f"Le document '{self.Titre}' est déjà retourné.")


class Livre(Document):
    def __init__(self,Titre,Auteur,Annee,ISBN, Nombre_de_pages):
        super().__init__(Titre,Auteur,Annee,ISBN)
        self.Nombre_de_pages = Nombre_de_pages

class Magazine(Document):
    def __init__(self,Titre,Auteur,Annee,ISBN, Numero_dedition,Periodicite):
        super().__init__(Titre,Auteur,Annee,ISBN)
        self.Numero_dedition = Numero_dedition
        self.Periodicite = Periodicite

class Bande_dessinee(Document):
    def __init__(self,Titre,Auteur,Annee,ISBN, Dessinateur,Couleur):
        super().__init__(Titre,Auteur,Annee,ISBN)
        self.Numero_dedition = Dessinateur
        self.Periodicite = Couleur
```

Code de test

```python
# ----------------------------------------------------------
# TEST
# ----------------------------------------------------------

# Création des documents
livre1 = Livre("Python avancé", "Dupont", 2022, "ISBN001", 350)
mag1 = Magazine("Science", "Martin", 2023, "ISBN002", 45, "Mensuelle")
bd1 = Bande_dessinee("Aventure Max", "Leroy", 2020, "ISBN003", "Durand", True)

documents = [livre1, mag1, bd1]

print("\n--- AFFICHAGE ---")
for doc in documents:
    doc.afficher()
    print()

print("\n--- TEST EMPRUNT ---")
livre1.emprunter()
livre1.emprunter()  # déjà emprunté

print("\n--- TEST RETOUR ---")
livre1.retourner()
livre1.retourner()  # déjà retourné


```

Résultat obtenu (exécution)

```
--- AFFICHAGE ---
Titre : Python avancé
Auteur : Dupont
Année : 2022
ISBN : ISBN001

Titre : Science
Auteur : Martin
Année : 2023
ISBN : ISBN002

Titre : Aventure Max
Auteur : Leroy
Année : 2020
ISBN : ISBN003


--- TEST EMPRUNT ---
Le document 'Python avancé' a été bien emprunté.
Le document 'Python avancé' est déjà emprunté.

--- TEST RETOUR ---
Le document 'Python avancé' a été bien retourné.
Le document 'Python avancé' est déjà retourné.
```

Conclusion

Ce TP a permis de maîtriser les principes fondamentaux de la programmation orientée objet en Python. Les exercices ont mis en lumière l'importance de la modularité, de l'héritage, et du polymorphisme dans la conception d'applications robustes. Le travail réalisé développe des compétences essentielles pour la modélisation orientée objet dans des systèmes complexes.