Atelier_4_annexe

N'oubliez pas d'utiliser des commentaires pour expliquer le fonctionnement de chaque partie de votre code.

Activité 1

Créez une classe de base Personne qui contient des informations de base sur une personne. Ensuite, créez une classe Client qui hérite de Personne et ajoute des détails spécifiques aux clients du restaurant. Enfin, créez une classe Reservation qui permet de réserver une table pour un certain nombre de personnes à une date et une heure spécifiques.

Solution :

```python
from datetime import datetime

# Classe de base pour une personne
class Personne:
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom

# Classe Client héritant de Personne
class Client(Personne):
    def __init__(self, nom, prenom, email):
        super().__init__(nom, prenom)
        self.email = email

# Classe Réservation
class Reservation:
    def __init__(self, client, date_heure, nb_personnes):
        self.client = client
        self.date_heure = date_heure
        self.nb_personnes = nb_personnes

    def afficher_reservation(self):
        print(f"Réservation au nom de {self.client.prenom} {self.client.nom}")
        print(f"Email: {self.client.email}")
        print(f"Date et heure: {self.date_heure.strftime('%Y-%m-%d %H:%M')}")
        print(f"Nombre de personnes: {self.nb_personnes}")

# Fonction de test
def test_reservation():
    client = Client("Dupont", "Jean", "jean.dupont@example.com")
    date_heure = datetime(2023, 4, 15, 19, 30)
    reservation = Reservation(client, date_heure, 4)
```

```
    reservation.afficher_reservation()

test_reservation()
```

Activité 2

Créez une classe Table qui représente une table dans le restaurant avec un numéro et une capacité. Ensuite, créez une classe TableRonde et une classe TableCarree qui héritent de Table et qui ont des attributs supplémentaires spécifiques.

Solution :

```
# Classe Table
class Table:
    def __init__(self, numero, capacite):
        self.numero = numero
        self.capacite = capacite

# Classe TableRonde héritant de Table
class TableRonde(Table):
    def __init__(self, numero, capacite, diametre):
        super().__init__(numero, capacite)
        self.diametre = diametre

# Classe TableCarree héritant de Table
class TableCarree(Table):
    def __init__(self, numero, capacite, longueur):
        super().__init__(numero, capacite)
        self.longueur = longueur

# Fonction de test
def test_tables():
    table_ronde = TableRonde(1, 4, 1.2)
    table_carree = TableCarree(2, 2, 0.8)
    print(f"Table ronde numéro {table_ronde.numero} pour {table_ronde.capa-
cite} personnes, diamètre {table_ronde.diametre}m.")
    print(f"Table carrée numéro {table_carree.numero} pour {table_carree.capa-
cite} personnes, longueur {table_carree.longueur}m.")

# Exécution de la fonction de test
test_tables()
```

Activité 3 :

Créez une classe Commande qui permet de gérer les commandes des clients. Ajoutez une classe Article qui représente un article du menu. Ensuite, créez une classe CommandeArticle qui hérite de Article et qui contient la quantité commandée.

Solution :

```
# Classe Client
class Client:
    def __init__(self, nom, prenom, email):
        self.nom = nom
        self.prenom = prenom
        self.email = email

# Classe Article
class Article:
    def __init__(self, nom, prix):
        self.nom = nom
        self.prix = prix

# Classe CommandeArticle héritant de Article
class CommandeArticle(Article):
    def __init__(self, nom, prix, quantite):
        super().__init__(nom, prix)
        self.quantite = quantite

# Classe Commande
class Commande:
    def __init__(self, client, articles):
        self.client = client
        self.articles = articles

    def calculer_total(self):
        return sum(article.prix * article.quantite for article in self.ar-
ticles)

# Fonction de test pour la commande
def test_commande():
    # Création d'un client
    client = Client("Dupont", "Jean", "jean.dupont@example.com")

    # Création de quelques articles
    article1 = CommandeArticle("Chaise", 50.0, 2)
    article2 = CommandeArticle("Table", 150.0, 1)
    article3 = CommandeArticle("Lampe", 20.0, 3)

    # Création d'une liste d'articles pour la commande
    articles = [article1, article2, article3]

    # Création de la commande
    commande = Commande(client, articles)

    # Calcul et affichage du total de la commande
    total = commande.calculer_total()
```

```
    print(f"Le total de la commande est : {total} €")

# Appel de la fonction de test
test_commande()
```

Combiner les trois codes :

```
from datetime import datetime

# Classe de base pour une personne
class Personne:
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom

# Classe Client héritant de Personne
class Client(Personne):
    def __init__(self, nom, prenom, email):
        super().__init__(nom, prenom)
        self.email = email

# Classe Réservation
class Reservation:
    def __init__(self, client, date_heure, nb_personnes):
        self.client = client
        self.date_heure = date_heure
        self.nb_personnes = nb_personnes

    def afficher_reservation(self):
        print(f"Réservation au nom de {self.client.prenom} {self.client.nom}")
        print(f"Email: {self.client.email}")
        print(f"Date et heure: {self.date_heure.strftime('%Y-%m-%d %H:%M')}")
        print(f"Nombre de personnes: {self.nb_personnes}")

# Classe Table
class Table:
    def __init__(self, numero, capacite):
        self.numero = numero
        self.capacite = capacite

# Classe TableRonde héritant de Table
class TableRonde(Table):
    def __init__(self, numero, capacite, diametre):
        super().__init__(numero, capacite)
        self.diametre = diametre

# Classe TableCarree héritant de Table
class TableCarree(Table):
```

```
    def __init__(self, numero, capacite, longueur):
        super().__init__(numero, capacite)
        self.longueur = longueur

# Classe Article
class Article:
    def __init__(self, nom, prix):
        self.nom = nom
        self.prix = prix

# Classe CommandeArticle héritant de Article
class CommandeArticle(Article):
    def __init__(self, nom, prix, quantite):
        super().__init__(nom, prix)
        self.quantite = quantite

# Classe Commande
class Commande:
    def __init__(self, client, articles):
        self.client = client
        self.articles = articles

    def calculer_total(self):
        return sum(article.prix * article.quantite for article in self.ar-
ticles)

# Classe principale Main
class Main:
    def test_reservation(self):
        client = Client("Dupont", "Jean", "jean.dupont@example.com")
        date_heure = datetime(2023, 4, 15, 19, 30)
        reservation = Reservation(client, date_heure, 4)
        reservation.afficher_reservation()

    def test_tables(self):
        table_ronde = TableRonde(1, 4, 1.2)
        table_carree = TableCarree(2, 2, 0.8)
        print(f"Table ronde numéro {table_ronde.numero} pour {table_ronde.ca-
pacite} personnes, diamètre {table_ronde.diametre}m.")
        print(f"Table carrée numéro {table_carree.numero} pour {table_car-
ree.capacite} personnes, longueur {table_carree.longueur}m.")

    def test_commande(self):
        client = Client("Dupont", "Jean", "jean.dupont@example.com")
        article1 = CommandeArticle("Chaise", 50.0, 2)
        article2 = CommandeArticle("Table", 150.0, 1)
        article3 = CommandeArticle("Lampe", 20.0, 3)
        articles = [article1, article2, article3]
        commande = Commande(client, articles)
```

```
        total = commande.calculer_total()
        print(f"Le total de la commande est : {total} €")

# Exécution des tests
if __name__ == "__main__":
    main = Main()
    main.test_reservation()
    main.test_tables()
    main.test_commande()
```

Activité 4

Créez une classe Personne avec des informations de base sur une personne. Ensuite, créez une classe Ouvrier qui hérite de Personne et ajoute des détails spécifiques aux ouvriers de l'usine, tels que l'identifiant de l'ouvrier et son poste. Enfin, créez une classe Absence qui enregistre les absences des ouvriers avec la date et la raison de l'absence.

Solution :

```
from datetime import date

# Classe de base pour une personne
class Personne:
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom

# Classe Ouvrier héritant de Personne
class Ouvrier(Personne):
    def __init__(self, nom, prenom, identifiant, poste):
        super().__init__(nom, prenom)
        self.identifiant = identifiant
        self.poste = poste

# Classe Absence
class Absence:
    def __init__(self, ouvrier, date_absence, raison):
        self.ouvrier = ouvrier
        self.date_absence = date_absence
        self.raison = raison

    def afficher_absence(self):
        print(f"Absence de {self.ouvrier.prenom} {self.ouvrier.nom}")
        print(f"Identifiant: {self.ouvrier.identifiant}")
        print(f"Date: {self.date_absence}")
```

```
        print(f"Raison: {self.raison}")

# Fonction de test
def test_absences():
    ouvrier = Ouvrier("Martin", "Luc", "O123", "Soudeur")
    absence = Absence(ouvrier, date(2023, 4, 10), "Rendez-vous médical")
    absence.afficher_absence()

# Exécution de la fonction de test
test_absences()
```

Activité 5

Créez une classe Horaire qui représente les horaires de travail d'un ouvrier avec un début et une fin de journée. Ensuite, créez une classe HoraireFlexible qui hérite de Horaire et permet à l'ouvrier d'avoir des horaires flexibles avec une plage horaire de disponibilité.

Solution :

```
# Classe Horaire
class Horaire:
    def __init__(self, debut, fin):
        self.debut = debut
        self.fin = fin

# Classe HoraireFlexible héritant de Horaire
class HoraireFlexible(Horaire):
    def __init__(self, debut, fin, plage_debut, plage_fin):
        super().__init__(debut, fin)
        self.plage_debut = plage_debut
        self.plage_fin = plage_fin

    def afficher_horaire(self):
        print(f"Horaire standard: de {self.debut} à {self.fin}")
        print(f"Plage horaire flexible: de {self.plage_debut} à
{self.plage_fin}")

# Fonction de test
def test_horaires():
    horaire_flexible = HoraireFlexible("08:00", "17:00", "07:00", "19:00")
    horaire_flexible.afficher_horaire()

# Exécution de la fonction de test
test_horaires()
```

Activité 6

Créez une classe Equipe qui représente une équipe d'ouvriers dans l'usine. Ajoutez une classe ChefEquipe qui hérite de Ouvrier et qui a la responsabilité de gérer les absences de son équipe. La classe Equipe doit contenir une liste d'ouvriers et une méthode pour ajouter des absences.

Solution :

```
from datetime import date

# Classe ChefEquipe
class ChefEquipe:
    def __init__(self, nom, prenom, identifiant, role):
        self.nom = nom
        self.prenom = prenom
        self.identifiant = identifiant
        self.role = role
        self.absences = []

    def ajouter_absence(self, ouvrier, date_absence, raison):
        self.absences.append((ouvrier, date_absence, raison))

    def afficher_absences(self):
        for absence in self.absences:
            ouvrier, date_absence, raison = absence
            print(f"{ouvrier.prenom} {ouvrier.nom} était absent le {date_ab-
sence} pour {raison}.")

# Classe Ouvrier
class Ouvrier:
    def __init__(self, nom, prenom, identifiant, role):
        self.nom = nom
        self.prenom = prenom
        self.identifiant = identifiant
        self.role = role

# Classe Equipe
class Equipe:
    def __init__(self, chef_equipe):
        self.chef_equipe = chef_equipe
        self.ouvriers = []

    def ajouter_ouvrier(self, ouvrier):
        self.ouvriers.append(ouvrier)

    def enregistrer_absence(self, ouvrier, date_absence, raison):
        if ouvrier in self.ouvriers:
            self.chef_equipe.ajouter_absence(ouvrier, date_absence, raison)
        else:
```

```
            print("L'ouvrier n'appartient pas à cette équipe.")

# Fonction de test
def test_gestion_equipes():
    chef = ChefEquipe("Durand", "Alice", "C456", "Chef d'équipe")
    equipe = Equipe(chef)

    ouvrier1 = Ouvrier("Petit", "Marc", "O789", "Opérateur machine")
    ouvrier2 = Ouvrier("Lefebvre", "Julie", "O101", "Assembleur")

    equipe.ajouter_ouvrier(ouvrier1)
    equipe.ajouter_ouvrier(ouvrier2)

    equipe.enregistrer_absence(ouvrier1, date(2023, 4, 12), "Maladie")
    equipe.enregistrer_absence(ouvrier2, date(2023, 4, 13), "Congé familial")

    chef.afficher_absences()

# Exécution de la fonction de test
test_gestion_equipes()
```