

Séries d'exercices POO

Exercice 1

Objectif : modéliser une petite bibliothèque dans laquelle les utilisateurs peuvent emprunter, rendre et rechercher des livres.

Classes :

Livre : représente les livres individuels de la bibliothèque.

Attributs : titre, auteur, isbn, nbrExemplaire.

Méthodes : afficher_details().

Membre : représente un membre de la bibliothèque qui peut emprunter des livres.

Attributs : nom, member_id, livres_empruntes (liste).

Méthodes : emprunter_livre(livre), afficher_liste_emprunts().

Bibliothèque : gère la collection de livres et de membres.

Attributs : livres (liste d'objets Livre), membres (liste d'objets Membre).

Méthodes :

ajouter_livre(livre) : ajoute un nouveau livre à la bibliothèque.

supprimer_livre(livre) : supprime un livre de la bibliothèque.

enregistrer_membre(membre) : enregistre un nouveau membre de la bibliothèque.

chercher_livre_par_titre(titre) : recherche un livre par titre.

emprunter_livre(membre, livre) : prêter un livre d'un membre.

retourner_livre (membre, livre) : retourner un livre d'un membre.

Exercice 2

Objectif : simuler un système bancaire avec différents types de comptes et autoriser les transactions entre comptes.

Classes :

Compte (classe de base) : classe de base pour tous les comptes bancaires.

Attributs : numero_compte, solde.

Méthodes : déposer(montant), retrait(montant), afficher_solde().

CheckingAccount (hérite de Compte) : compte courant avec frais de transaction.

Attributs supplémentaires : frais_transaction.

Méthodes : remplace le retrait pour déduire les frais de transaction pour chaque retrait (2%).

SavingsAccount (hérite de Compte) : compte d'épargne avec intérêts.

Attributs supplémentaires : taux_interet.

Méthodes : appliquer_interet() : augmente le solde en fonction du taux d'intérêt.

Client : représente un client bancaire avec plusieurs comptes.

Attributs : client_id, nom, comptes (liste des types de comptes).

Méthodes : ouvrir_compte(type_compte), fermer_compte(type_compte), afficher_comptes().

Banque : gère tous les clients et toutes les transactions.

Attributs : clients (liste des objets Client).

Méthodes :

ajouter_client(client), supprimer_client(client).

transférer_fonds(compte_emetteur, compte_destination, amount) : transfère des fonds entre les comptes.

Exercices 3

Objectif : créer un système dans lequel différents types de repas ont des méthodes de préparation et de tarification uniques.

Classes :

Repas (classe de base) :

Attributs encapsulés : nom, cout.

Méthodes :

preparer() : Imprime un message générique, « Préparation du repas ».

calculer_prix() : Renvoie cout.

Entree (hérite de Repas) :

Attributs encapsulés : taille_portion.

Méthodes remplacées :

preparer() : Imprime « Préparation de l'apéritif : <nom>, taille de la portion : < taille_portion >.»

calculer_prix() : Ajoute 5 % à cout pour les grandes portions.

PlatPrincipal (hérite de Repas) :

Attributs encapsulés : cuisine_type.

Méthodes remplacées :

preparer() : Imprime « Cuisson du plat principal : <nom> dans le style <cuisine_type>. »

calculer_prix() : ajoute 10 % à cout si cuisine_type est « international ».

Dessert (hérite de Repas) :

Attributs encapsulés : sans_sucre.

Méthodes remplacées :

preparer() : affiche « Préparation du dessert : <nom>, sans sucre » si sans_sucre est True, sinon « Préparation du dessert : <nom>, avec sucre ».

calculate_price() : réduit cout de 10 % si sans_sucre est True.

Créez des instances pour différents types de repas et appelez preparer() et calculer_prix() pour observer différentes méthodes de préparation et de tarification. Utilisez le polymorphisme en stockant les repas dans une liste et en appelant preparer() et calculer_prix() sur chacun d'eux dans une boucle.

Exercice 4

Objectif : créer un système d'adhésion dans lequel différents types de membres de la bibliothèque ont des limites d'emprunt et des frais uniques.

Classes :

Membre (classe de base) :

Attributs encapsulés : nom, frais_adhesion.

Méthodes :

calculer_frais_annuels() : renvoie frais_adhesion.

limit_emprunt() : renvoie une limite d'emprunt générique, par exemple 3 livres.

AdultMembre (hérite de Membre) :

Attributs encapsulés : frais_supplementaire.

Méthodes remplacées :

calculer_frais_annuels() : renvoie frais_adhesion + frais_supplementaire.

limit_emprunt() : augmente la limite d'emprunt à 10 livres.

EnfantMembre (hérite de Membre) :

Méthodes remplacées :

calculer_frais_annuels() : renvoie la moitié de frais_adhesion.

limit_emprunt() : définit la limite d'emprunt à 5 livres et affiche « Autorisation parentale requise ».

SeniorMembre (hérite de Membre) :

Attributs encapsulés : pourcentage_rabais.

Méthodes remplacées :

calculer_frais_annuels() : applique une remise en soustrayant pourcentage_rabais de frais_adhesion.

limit_emprunt() : définit la limite d'emprunt à 7 livres.

Démontrer l'encapsulation en gardant les attributs privés et en autorisant l'accès via des méthodes.

Créer des instances de chaque type de membre, en montrant les différences dans les limites d'emprunt et les frais.

Utiliser le polymorphisme en stockant les membres dans une liste et en itérant pour appeler limit_emprunt() et calculer_frais_annuels() sur chacun.