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) : retorune 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 : deposer(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).

transferer_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.

Exercice 5

Objectif : Concevoir un système pour gérer les projections de films, les réservations de sièges et les ventes de billets dans un cinéma.

Classes :

Film :

Attributs (encapsulés) : _titre, _realisateur, _duree_minutes.

Méthodes :

afficher_details() : Affiche le titre, le réalisateur et la durée du film.

Salle :

Attributs (encapsulés) : _numero_salle, _capacite, _sieges (liste des objets Siege).

Méthodes :

afficher_salle() : Affiche le numéro et la capacité de la salle.

generer_sieges() : Crée les objets Siege en fonction de la capacité.

afficher_sieges_disponibles() : Affiche les sièges disponibles.

Siege :

Attributs (encapsulés) : _numero_siege, _disponible (booléen).

Méthodes :

afficher_status() : Affiche si le siège est disponible ou réservé.

reserver() : Change l'état en non disponible.

liberer() : Change l'état en disponible.

Projection (hérite de Film) :

Attribut supplémentaire : _salle (objet Salle), _heure.

Méthodes :

afficher_details() : Affiche les détails spécifiques de la projection.

reserver_siege(numero_siege) : Réserve un siège spécifique si disponible.

afficher_sieges_reserves() : Affiche tous les sièges réservés.

Billet :

Attributs (encapsulés) : _id_billet, _projection, _siege, _prix.

Méthodes :

afficher_billet() : Affiche les détails du billet.

Client :

Attributs (encapsulés) : _nom, _id_client, _billets (liste des objets Billet).

Méthodes :

acheter_billet(projection, numero_siege, prix) : Achète un billet pour une projection spécifique.

afficher_billets() : Affiche tous les billets achetés par le client.

Cinema :

Attributs : _films (liste des objets Film), _salles (liste des objets Salle), _projections (liste des objets Projection), _clients (liste des objets Client).

Méthodes :

ajouter_film(film), ajouter_salle(salle), programmer_projection(projection), enregistrer_client(client).

afficher_programme() : Affiche toutes les projections programmées.

afficher_clients() : Affiche tous les clients inscrits.

- Instanciez un objet Cinema.
- Créez plusieurs objets Film et Salle avec des données appropriées.
- Ajoutez ces films et salles au cinéma en utilisant ajouter_film et ajouter_salle.
- Créez des objets Projection en associant un film, une salle et une heure.
- Ajoutez ces projections au cinéma en utilisant programmer_projection.
- Générez les sièges pour chaque salle en utilisant generer_sieges.
- Créez plusieurs objets Client.
- Ajoutez ces clients au cinéma en utilisant enregistrer_client.
- Un client choisit une projection et un siège disponible.
- Utilisez reserver_siege pour réserver le siège dans la projection.
- Créez un objet Billet avec les détails de la réservation.
- Ajoutez le billet à la liste des billets du client en utilisant acheter_billet.
- Utilisez afficher_programme pour voir toutes les projections disponibles.
- Utilisez afficher_sieges_disponibles pour une projection spécifique.
- Utilisez afficher_billets pour voir les billets achetés par un client.
- Utilisez afficher_clients pour voir tous les clients inscrits.