TP2 : POO avec Java

Exercice 1 : Gestion d'une Société de Vente

Contexte : Une société de vente en ligne gère différents types de produits. Chaque produit a des caractéristiques communes mais aussi des spécificités.

Créez une classe Produit avec :

1. Les **attributs** suivants :

- **reference (String)** : référence unique du produit
- **nom (String)** : nom du produit
- **prix (double)** : prix en euros
- **stock (int)** : quantité en stock

2. Un **constructeur** qui initialise **tous les attributs**
3. Les **getters** et **setters** nécessaires
4. Une méthode **afficherInfos()** qui affiche toutes les informations du produit

Créez une classe Livre qui hérite de Produit avec :

1. Les **attributs** supplémentaires :

- **auteur (String)** : nom de l'auteur
- **isbn (String)** : numéro ISBN

2. Un **constructeur** qui initialise **tous les attributs** (y compris ceux de la classe parent)
3. Une **redéfinition** de la méthode **afficherInfos()** pour inclure les nouvelles informations

Dans une classe Main :

1. Créez un Produit et un Livre
2. Affichez les informations des deux objets
3. Testez si un Livre est bien une instance de Produit (**instanceOf**)

Questions bonus :

1. Que se passe-t-il si vous essayez d'accéder directement aux attributs de Produit depuis la classe Livre ?
2. Comment devriez-vous modifier les attributs de Produit pour qu'ils soient accessibles dans Livre ?

Exercice 2 : Système de Gestion d'une Médiathèque

Contexte : Une médiathèque souhaite informatiser sa gestion des emprunts. Elle propose différents types de médias (livres, DVDs) avec des règles de prêt spécifiques à chaque type.

Créez une classe parent Media avec :

1. Les attributs suivants :

- **reference (final String)** : référence unique du média
- **titre (protected String)**
- **disponible (protected boolean)**
- **static int nombreMedias** : pour générer les références

2. Un **constructeur** qui initialise :

- Le **titre**
- Génère automatiquement la **référence** (exemple : "MEDIA_1", "MEDIA_2", etc.)
- Initialise disponible à **true**

3. Les méthodes suivantes :

- **emprunter() :** change l'état de disponibilité
- **rendre() :** remet le média comme disponible
- **calculerAmende(int joursRetard) :** version de base du calcul d'amende
- **isDisponible() :** retourne l'état de disponibilité

Créez une classe Livre qui hérite de Media :

1. Attributs supplémentaires :

- **auteur (private String)**
- **amendeParJour (private double)**

2. Redéfinissez :

- **calculerAmende(int joursRetard)** : retourne joursRetard * amendeParJour
- **toString()** : affiche les informations du livre

Créez une classe DVD qui hérite de Media :

1. Attributs supplémentaires :

- **realisateur (private String)**
- **amendeParJour (private double)**

2. Redéfinissez :

- **calculerAmende(int joursRetard)** :
  1. **Si joursRetard > 7 :** (amendeParJour * 2) * joursRetard
  2. **Sinon :** amendeParJour * joursRetard

Créez une classe Adherent :

1. Attributs :

- **numero (private String)**
- **nom (private String)**
- **emprunts (private Media[])** : tableau simple de taille 5 maximum

2. Méthodes :

- **emprunterMedia(Media media) :** ajoute un média au tableau d'emprunts
- **rendreMedia(Media media) :** retire un média du tableau d'emprunts
- **calculerAmendes(int joursRetard) :** calcule l'amende totale

Dans le main, testez :

1. Créez un adhérent
2. Créez plusieurs livres et DVDs
3. Effectuez des opérations :
- Emprunts de différents médias
- Retours avec calcul d'amendes
- Tentative d'emprunt quand le tableau est plein

Questions bonus :

1. Comment le polymorphisme nous aide-t-il à gérer différents types de médias ?
2. Que se passe-t-il si on essaie d'emprunter un média déjà emprunté ?
3. Comment pourrait-on améliorer la gestion du tableau d'emprunts ?

Exercice 3 : Gestion des Comptes Bancaires

Contexte : Une banque propose différents types de comptes bancaires. Chaque type de compte a ses propres règles de calcul d'intérêts et de frais.

Créez une classe abstraite CompteBancaire avec :

1. Les **attributs** suivants :

- **numeroCompte (final String)** : numéro unique du compte

- **solde (protected double)**
- **titulaire (private String)**
- **static int nombreComptes** : pour générer les numéros de compte

2. Un **constructeur** qui initialise :

- le **titulaire**
- génère automatiquement le **numéro de compte** (exemple : "COMPTE_1", "COMPTE_2", etc.)

3. Les **méthodes** suivantes :

- **depot(double montant)** : final, ajoute le montant au solde
- **retrait(double montant)** : peut être redéfinie, retire le montant du solde
- **abstract calculerInterets()** : calcule les intérêts selon le type de compte
- **getSolde()** : retourne le solde actuel

Créez une classe CompteEpargne qui hérite de CompteBancaire :

1. Attribut supplémentaire :

tauxInteret (private final double)

2. Redéfinissez :

- la méthode calculerInterets() : retourne solde * tauxInteret
- la méthode retrait() : doit vérifier que le solde après retrait reste positif

Créez une classe CompteCourant qui hérite de CompteBancaire :

1. Attributs supplémentaires :

- **decouvertAutorise (private final double)**
- **tauxInteretDecouvert (private final double)**

2. Redéfinissez :

- **la méthode calculerInterets() :**

  1. **Si solde > 0 :** pas d'intérêts
  2. **Si solde < 0 :** solde * tauxInteretDecouvert

- **la méthode retrait() :** vérifie que le découvert maximum n'est pas dépassé

Créez une classe Client :

1. Attributs :

- **nom (private final String)**
- **comptes (private ArrayList<CompteBancaire>)**

2. Méthodes :

- **ajouterCompte(CompteBancaire compte)**
- **getSoldeTotal()** : somme des soldes de tous les comptes
- **calculerInteretsTotal()** : somme des intérêts de tous les comptes

Dans le main, testez :

1. Créez un client avec un compte épargne et un compte courant
2. Effectuez des opérations :

- Dépôts
- Retraits (certains doivent échouer)
- Calcul des intérêts

3. Affichez le solde total du client

Questions bonus :

1. Pourquoi la méthode depot() est-elle déclarée final ?
2. Comment gérer le cas où un retrait dépasse le découvert autorisé ?
3. Que se passerait-il si on voulait ajouter un nouveau type de compte ?

Exercice 4 : Gestion d'un Magasin de Produits

Contexte : On veut créer un système pour gérer différents types de produits dans un magasin, avec la possibilité d'appliquer différentes réductions.

Créez une interface Remisable avec les méthodes suivantes :

- **double calculerPrixRemise()** : calcule le prix après remise
- **boolean estEnPromotion()** : vérifie si le produit est en promotion

Créez une classe abstraite Produit qui implémente Remisable :

1. Attributs :

- **private final String** reference
- **private String** nom
- **protected double** prix
- **private boolean** promotion

2. Méthodes **abstraites** :

- **public abstract** String getDescription()

Créez deux classes qui héritent de Produit :

1. Classe ProduitAlimentaire :

- **Attribut supplémentaire :** private LocalDate dateExpiration
- La remise est de **20%** si le **produit est en promotion**
- La remise est de **50%** si la **date d'expiration est dans moins de 7 jours**

2. Classe ProduitElectronique :

- **Attribut supplémentaire :** private int garantieAnnees
- La remise est de **10%** si le **produit est en promotion**
- Ajoute **50€** au prix si la **garantie est > 2 ans**

Questions :

1. Implémentez toutes les classes en respectant l'encapsulation
2. Comment la méthode calculerPrixRemise() sera différente dans chaque classe ?
3. Dans le main, créez quelques produits et testez les différentes remises