TD 4 - POO

Partie 1 : Classes Abstraites

Contexte

Vous devez modéliser un système de gestion d'instruments de musique pour un magasin.

Exercice 1 : Hiérarchie d'instruments

Objectif : Créer une hiérarchie de classes avec une classe abstraite **InstrumentMusique.**

Créer une classe abstraite InstrumentMusique avec les caractéristiques suivantes :

Attributs protégés :

- String nom : le nom de l'instrument
- String type : la catégorie ("Cordes", "Vent", "Percussion")
- double prix : le prix HT en euros

Constructeur :

- Initialise tous les attributs

Méthodes abstraites :

- void jouer() : doit afficher un son ou une description spécifique à l'instrument
- void accorder() : doit afficher comment accorder l'instrument

Méthodes concrètes :

- void afficherInfo() : affiche le nom, type et prix HT de l'instrument
- double calculerPrixAvecTVA() : retourne le prix TTC (TVA = 20%)

Ensuite, créer trois classes concrètes qui héritent de InstrumentMusique :

Classe 1 : Guitare

- Ajouter un attribut int nombreCordes
- Implémenter jouer() pour produire un son caractéristique de guitare
- Implémenter accorder() pour expliquer comment accorder ses cordes

Classe 2 : Piano

- Ajouter un attribut int nombreTouches
- Implémenter jouer() pour produire un son caractéristique de piano
- Implémenter accorder() pour expliquer que c'est une opération professionnelle

Classe 3 : Batterie

- Ajouter un attribut `int nombreFuts`
- Implémenter `jouer()` pour produire un son caractéristique de batterie
- Implémenter `accorder()` pour expliquer l'ajustement des peaux

Test

Écrire une classe `TestInstruments` qui :

1. Créer un tableau contenant au moins une guitare, un piano et une batterie
2. Pour chaque instrument, affiche ses informations, le fait jouer, l'accorde et affiche son prix TTC

Partie 2 : Interfaces

Contexte

Vous devez créer un système de gestion d'appareils électroniques avec différentes capacités : être rechargeable, portable, ou connecté à Internet.

Exercice 2 : Système d'appareils électroniques

Partie A : Définir les interfaces

Créer trois interfaces :

Interface 1 : Rechargeable

- Méthode `void recharger()`
- Méthode `int getNiveauBatterie()`
- Méthode `void afficherBatterie()`

Interface 2 : Portable

- Méthode `double getPoids()` retournant le poids en kg
- Méthode `void afficherPortabilite()`

Interface 3 : Connecte

- Méthode `void connecterWifi(String reseau)`
- Méthode `void deconnecter()`
- Méthode `boolean estConnecte()`

Partie B : Classe abstraite AppareilElectronique

Créer une classe abstraite AppareilElectronique :

Attributs protégés :

- String marque
- String modele
- double prix

Constructeur :

- Initialise tous les attributs

Méthode abstraite :

- void allumer()

Méthode concrète :

- void afficherCaracteristiques() : affiche marque, modèle et prix

Partie C : Classes concrètes

Créer trois classes qui héritent de AppareilElectronique et implémentent les interfaces appropriées :

Classe 1 : Smartphone

- Implémente les trois interfaces : Rechargeable, Portable, Connecte
- Ajouter un attribut pour la batterie (0-100)
- Ajouter un attribut pour le poids
- Ajouter un attribut pour savoir si le WiFi est actif
- Ajouter un attribut pour le nom du réseau WiFi actuel
- Implémenter toutes les méthodes des interfaces
- Implémenter allumer() pour afficher un message personnalisé

Classe 2 : OrdinateurPortable

- Implémente les trois interfaces : Rechargeable, Portable, Connecte
- Même structure que Smartphone mais avec des valeurs différentes (poids plus élevé, batterie plus grande)
- Implémenter les méthodes avec des messages différents

Classe 3 : Tablette

- Implémente deux interfaces : Rechargeable, Portable (pas de WiFi pour cette version)
- Ajouter un attribut pour la batterie et le poids
- Implémenter les méthodes de ces deux interfaces
- Implémenter allumer() avec un message personnalisé

Test

Écrire une classe TestAppareils qui :

1. Crée un smartphone, un ordinateur portable et une tablette
2. Pour chaque appareil :
    - Affiche ses caractéristiques
    - L'allume
    - Utilise toutes ses capacités (recharger, afficher batterie, connecter WiFi si possible)
    - Affiche l'état final

Créer une méthode `static` qui prend en paramètre n'importe quel appareil `Rechargeable` et effectue un cycle complet de recharge

Partie 3 : Polymorphisme Avancé

Contexte

Vous devez créer un système de paiement flexible pour un site e-commerce qui gère différents moyens de paiement avec des frais variables.

Exercice 3 : Système de paiement polymorphe

Partie A : Classe abstraite MoyenPaiement

Créer une classe abstraite `MoyenPaiement` :

Attributs protégés :

- `String titulaire` : nom du propriétaire du moyen de paiement
- `double solde` : montant disponible

Constructeur :

- Initialise ces attributs

Méthodes abstraites :

- `boolean payer(double montant)` : effectue un paiement et retourne true si succès, false si solde insuffisant
- `String getTypePaiement()` : retourne le type de moyen de paiement

Méthodes concrètes :

- `void afficherSolde()` : affiche le solde actuel
- `boolean verifierSolde(double montant)` : retourne true si solde >= montant

Méthodes surchargées (overloading) :
Créer trois versions de la méthode `effectuerTransaction` :

1. Une version avec seulement le montant
2. Une version avec le montant et une description

3. Une version avec le montant, description et commission

Partie B : Classes concrètes

Créer trois classes qui héritent de MoyenPaiement :

Classe 1 : CarteBancaire

- Ajouter un attribut pour stocker le numéro de carte (afficher seulement les 4 derniers chiffres pour sécurité)
- Implémenter payer() :
    - Vérifier si le solde est suffisant
    - Si oui : déduire le montant et retourner true
    - Si non : afficher un message d'erreur et retourner false
- Implémenter getTypePaiement() : retourner "Carte Bancaire"

Classe 2 : PayPal

- Ajouter un attribut pour l'adresse email
- Implémenter payer() :
    - Ajouter une commission de 2% au montant
    - Vérifier si le solde couvre montant + commission
    - Si oui : déduire le tout et retourner true
    - Si non : afficher message d'erreur et retourner false
- Implémenter getTypePaiement() : retourner "PayPal"

Classe 3 : Crypto

- Ajouter un attribut pour l'adresse de portefeuille (wallet)
- Implémenter payer() :
    - Ajouter une commission de 1% au montant
    - Vérifier le solde
    - Si oui : déduire montant + commission, afficher message de transaction blockchain, retourner true
    - Si non : retourner false
- Implémenter getTypePaiement() : retourner "Cryptomonnaie"

Test

Écrire une classe TestPaiement qui :

1. Crée un tableau polymorphe contenant une carte bancaire, un compte PayPal et un portefeuille crypto
2. Pour chaque moyen :
    - Effectue un paiement simple (utilisant la première version surchargée)
    - Effectue un paiement avec description (deuxième version)
    - Effectue un paiement avec commission (troisième version)

- Affiche le solde final

Partie 4 : Casting et instanceof

Contexte

Vous devez créer un système de gestion de parc informatique qui doit identifier le type exact de chaque équipement et effectuer des opérations spécifiques.

Exercice 4 : Gestion de parc informatique

Partie A : Hiérarchie de base

Créer une classe abstraite `Equipement` :

Attributs protégés :

- `String reference` : code unique de l'équipement
- `String emplacement` : où l'équipement est situé
- `boolean enService` : si l'équipement fonctionne

Constructeur :

- Initialise ces attributs

Méthode abstraite :

- `void effectuerMaintenance()` : affiche les actions de maintenance

Méthode concrète :

- `void afficherEtat()` : affiche la référence, l'emplacement et l'état du service

Créer trois classes filles :

Classe 1 : Ordinateur

- Ajouter un attribut `String systemeExploitation`
- Créer une méthode spécifique `void redemarrer()`
- Implémenter `effectuerMaintenance()` pour décrire la maintenance d'un ordinateur

Classe 2 : Imprimante

- Ajouter un attribut `int niveauEncre` (0-100)
- Créer une méthode spécifique `void remplacerCartouche()`
- Implémenter `effectuerMaintenance()` pour décrire la maintenance d'une imprimante

Classe 3 : Serveur

- Ajouter un attribut `int nombreCPU`

- Créer une méthode spécifique `void sauvegarder()`
- Implémenter `effectuerMaintenance()` pour décrire la maintenance d'un serveur

Partie B : Méthode avec casting sécurisé

Créer une classe `GestionParc` avec une méthode statique `diagnostiquerEquipement(Equipement eq)` qui :

1. Affiche l'état de l'équipement
2. Effectue la maintenance
3. Utilise `instanceof` pour identifier le type exact :
    - Si c'est un Ordinateur : afficher l'OS et redémarrer
    - Si c'est une Imprimante : afficher le niveau d'encre et remplacer la cartouche si niveau < 20%
    - Si c'est un Serveur : afficher le nombre de CPUs et effectuer une sauvegarde

Test

Écrire une classe `TestParc` qui :

1. Crée un tableau `Equipement[]` contenant au moins 5 équipements de types variés
2. Parcourt le tableau et appelle `diagnostiquerEquipement()` pour chaque élément
3. Affiche les statistiques finales :
    - Nombre d'ordinateurs
    - Nombre d'imprimantes
    - Nombre de serveurs
    - Total d'équipements