TP Avancé sur le POO en PHP

Exercice 1

Tu es chargé de créer un système de gestion de produits pour une boutique en ligne. Chaque produit possède un **nom**, un **prix**, et un **code produit unique**. Le nom et le prix doivent être encapsulés (privés), tandis que le code produit ne peut être défini qu'une seule fois.

Tu dois aussi permettre:

- De modifier le prix via un setter
- D'afficher proprement les détails d'un produit avec **__toString()**
- De détecter dynamiquement des propriétés avec **__get et __isset.**

Questions

1. **Création de la classe :**
   - Déclare une classe Produit avec trois propriétés : **$nom, $prix, $codeProduit**.
   - **$codeProduit** doit être en readonly.
2. Implémente un constructeur pour initialiser les trois propriétés.
3. Rends **$nom** et **$prix private**, puis crée les méthodes **getNom(), getPrix()** et **setPrix()**.
4. Implémente la méthode **__toString()** pour retourner une phrase du type :
   "Produit: $nom | Prix: $prix € | Code: $codeProduit"
5. Accès dynamique avec __get() et __isset() :
   - Implémente **__get()** pour retourner les propriétés privées en accès indirect.
   - Implémente **__isset()** pour savoir si une propriété est définie.
6. **Instanciation et tes**t :
   - Crée un objet Produit, affecte-lui un nom, prix et code.
   - Teste l'affichage automatique (**echo $produit**) et **isset($produit->nom).**

Exercice 2

Tu dois créer une mini-application orientée objet pour gérer une **bibliothèque**. Elle contiendra plusieurs **livres**, chacun avec un **titre**, un **auteur** et un **nombre de pages**.
Une classe **Bibliotheque** permettra d'ajouter des livres, de les afficher et de faire des recherches.

Questions

1. **Créer la classe Livre :**

   - Définis les propriétés **titre**, **auteur**, **nbPages** (toutes **private**).
   - Ajoute un **constructeur** et des **getters**.

2. **Créer la classe Bibliotheque:**

   - Elle doit contenir **un tableau de livres** (private array $livres = [];)

3. **Ajout de livre :**

   - Crée une méthode **ajouterLivre(Livre $livre)** qui ajoute un livre à la collection.

4. **Afficher tous les livres :**

   - Implémente une méthode **afficherLivres()** qui liste les livres avec un **foreach.**

5. **Rechercher un livre par titre :**

   - Crée une méthode **chercherParTitre(string $titre)** qui retourne un livre si le titre correspond

6. **Tester le tout:**

   - Crée une bibliothèque, ajoute plusieurs livres, puis affiche-les et cherches-en un

Exercice 3

On modélise un **système d'utilisateurs**. Il y a des Utilisateurs classiques, et des Admins qui héritent des utilisateurs mais ont un comportement légèrement différent à la connexion.

Questions

1. **Créer la classe Utilisateur :**

   - Ajoute une propriété **$nom**, un constructeur, et une méthode **seConnecter()** qui affiche "Utilisateur connecté".

2. **Créer la classe Admin:**

   - Fais-la hériter de Utilisateur
   - Redéfinis la méthode **seConnecter()** pour afficher "Admin connecté"

3. **Appel à parent:**

   - Dans la méthode de Admin, appelle aussi la méthode parent avec **parent::seConnecter()**

4. **Test du comportment:**
    - Crée un objet **utilisateur** et un objet **admin**, puis appelle **seConnecter()** sur les deux
5. **Ajouter une méthode spécifique à l'admin :**
    - Crée une méthode **supprimerUtilisateur(Utilisateur $u)** uniquement pour l'admin
    - Il va tout simplement afficher : « Suppression de l'utilisateur $nom »
6. **Ajouter un tableau d'utilisateurs et tester la méthode admin :**
    - Ajoute quelques utilisateurs dans un tableau et appelle **supprimerUtilisateur()** sur un admin fictif

Exercice 4

Tu vas créer un système simulant les **capacités des banques**.
Les banques peuvent effectuer des **retraits** et des **virements**.

Tu vas modéliser ces fonctionnalités à l'aide de **deux interfaces** et intégrer un **trait Logger** pour enregistrer les actions effectuées.

Questions:

1. **Créer deux interfaces Retrait et Virement :**
    - Chaque interface doit déclarer une méthode : retirer() ou virement().
2. **Créer la classe BanqueGenerale:**
    - Elle doit implémenter **les deux interfaces** et définir les méthodes associées.
3. **Créer un trait Logger:**
    - Le trait doit contenir une méthode log($message) qui affiche ou enregistre un message.
4. **Utiliser le trait dans la classe BanqueGenerale :**
    - Lorsqu'une opération de retrait ou de virement est effectuée, journaliser l'action avec le trait.
5. **Créer d'autres classes de banques :**
    - Par exemple, BanqueRetrait (implémente seulement Retrait) et BanqueVirement (implémente seulement Virement).
6. **Créer un tableau contenant différentes banques et les faire agir dynamiquement :**

- Utilise une boucle foreach, teste si la banque est instanceof Retrait ou Virement, puis appelle la méthode correspondante.

Exercice 5

Tu veux un objet "flexible" qui puisse stocker dynamiquement n'importe quelles données sans avoir besoin de les définir à l'avance.
Tu vas utiliser les méthodes magiques **__get, __set, __isset, __unset** pour créer un container de données dynamique.

Questions :

1. **Créer la classe DataContainer:**
    - Ajoute une propriété privée $data = [] pour stocker les données dynamiques
2. **Implémente __set($name, $value):**
    - Elle doit ajouter la valeur dans le tableau $data sous la clé $name
3. **Implémente __get($name) :**
    - Elle doit retourner la valeur associée dans $data ou null si non défini
4. **Implémente __isset($name) et __unset($name) :**
    - __isset() doit vérifier si la clé existe, __unset() la supprimer
5. **Créer un objet, stocker des données dynamiquement :**
    - Ex : $container->nom = "Alice"; puis affiche $container->nom
6. **Test complet :**
    - Vérifie avec isset, unset, tente d'accéder à une clé inexistante