Formation PHP - Achref El Mouelhi

# TP 3 : Héritage, classes abstraite/finale et interface

## Énoncé

Nous voulons développer une application de gestion de compte pour une une nouvelle banque qui prévoit un démarrage d'activité en 2023. Pour ce faire, les concepteurs ont jugé nécessaire la séparation des opérations en deux interfaces : `OperationSimple` ayant les méthodes abstraites suivantes :

- `crediter(int $somme): void`
- `debiter(int $somme): boolean`

et `OperationAvancee` avec la méthode abstraite suivante :

- `virement(Compte $beneficiaire, int $somme): boolean`

À l'ouverture de ses portes, la banque envisage autoriser deux types de compte :

- compte courant et
- compte épargne.

Les deux comptes sont caractérisés par :

- un `$identifiant` (de type `int`) et
- un $solde (de type `float`).

Un compte courant aura de plus un attribut `$decouvert` (de type `float`) et un compte épargne aura un `$tauxInteret` (de type `float`). Dans tous les cas, un compte appartient à un seul client. La banque n'exclut pas la possibilité d'étendre son activité sur certains autres types de compte (compte conjoint, compte professionnel...).

Quel que soit le type de compte, nous devons permettre au titulaire de créditer ou débiter son compte. Cette dernière opération est possible si certaines conditions sont remplies :

- Pour un compte courant, on autorise l'opération si on ne dépasse pas la valeur de découvert après avoir débitée la somme souhaitée.
- Pour un compte épargne, on autorise l'opération si le solde ne devient pas négatif après avoir débitée la somme souhaitée.

Le virement n'est autorisé que depuis un compte courant. Le compte bénéficiaire peut être de type courant ou épargne. Dans tous les cas, on ne l'autorise que si on parvient à débiter la somme du compte émetteur.

Les méthodes `virement` et `debiter` retournent `true` en cas de réussite de l'opération, `false` en cas d'échec.

Pour chaque client, nous voulons enregistrer

- son `$identifiant` (de type `int`),
- son `$nom` (de type `String`) et
- son `$prénom` (de type `String`).

Un client de la banque peut avoir un compte courant, jusqu'à trois maximum, ainsi qu'un compte épargne. Quel que soit le nombre de ses comptes, nous devons lui permettre de les consulter.

# Questions

1. Élaborez un diagramme de classes. Construisez de la manière la plus optimale la hiérarchie des classes/interfaces. Prenez en compte les opérations autorisées pour chaque type de compte.
2. Créez les classes `Compte`, `CompteCourant` et `CompteEpargne` dans trois fichiers séparés. Générez / définissez les getters / setters et constructeur.
3. Implémentez la méthode `debiter()` dans les deux classes `CompteCourant` et `CompteEpargne`.
4. Quel que soit le type de compte, la méthode `crediter()` a une seule implémentation et ne changera pas lorsque la banque décidera de s'ouvrir sur d'autres types de compte. Proposez la meilleure implémentation de cette méthode.
5. Implémentez la méthode `virement()`.
6. La classe `Compte` doit forcer `CompteCourant` et `CompteEpargne` à avoir leur propre implémentation de la méthode `imprimer()` : cette méthode affiche toutes les caractéristiques d'un compte (tout sauf identifiant) ainsi que son type (courant ou épargne). Implémentez la méthode `imprimer` dans les deux classes `CompteCourant` et `CompteEpargne`.
7. Créez la classe `Client` et générez / définissez les getters / setters et constructeur.
8. Ajoutez l·es attribut·s nécessaire·s qui permettr·a·ont de déterminer le nombre de comptes courants d'un client.
9. Dans `Client`, définissez une méthode `ajouterCompteCourant()` qui permet d'ajouter un nouveau compte courant s et qui retourne `true` en cas de réussite, `false` en cas d'échec.
10. Dans `Client`, définissez une méthode `ajouterCompteEpragne()` qui permet d'ajouter un compte épargne et qui retourne `true` en cas de réussite, `false` en cas d'échec.
11. Définissez une méthode `consulter()` qui affiche les données du client ainsi que les détails de tous ses comptes.
12. Testez progressivement toutes les méthodes implémentées et les méthodes à implémenter dans `index.php`.
13. Dans `CompteEpargne`, écrivez une méthode `ajouterInteret()` qui ajoute les intérêts au solde en fonction de la valeur de l'attribut `tauxInteret`.
14. Faites les changements nécessaires pour permettre à un client de faire un virement depuis son compte épargne vers un de ses comptes courants (bien sûr si débiter son compte épargne est possible).