TP BANQUE ***SAGIM 01/04/2024***

TP en PHP qui met en œuvre les concepts d'héritage, d'encapsulation et d'interface dans le contexte d'une banque. Dans cet exemple, nous allons créer une hiérarchie de classes pour représenter différents types de comptes bancaires (compte d'épargne et compte courant) et utiliser des interfaces pour définir des comportements communs.

```php
<?php
// Interface pour les comptes bancaires
interface BankAccount {
    public function deposit($amount);
    public function withdraw($amount);
    public function getBalance();
}
// Classe de base pour les comptes bancaires
class BaseAccount implements BankAccount {
    protected $balance;
    public function __construct($balance = 0) {
        $this->balance = $balance;
    }
    public function deposit($amount) {
        $this->balance += $amount;
    }
    public function withdraw($amount) {
        if ($amount <= $this->balance) {
            $this->balance -= $amount;
        } else {
            echo "Solde insuffisant\n";
        }
    }

    public function getBalance() {
        return $this->balance;
    }
}
// Classe pour un compte d'épargne
class SavingsAccount extends BaseAccount {
    private $interestRate;
    public function __construct($balance = 0, $interestRate = 0.02) {
        parent::__construct($balance);
        $this->interestRate = $interestRate;
    }
    public function applyInterest() {
        $this->balance += $this->balance * $this->interestRate;
    }
}
// Classe pour un compte courant
class CheckingAccount extends BaseAccount {
    private $overdraftLimit;
    public function __construct($balance = 0, $overdraftLimit = -100) {
        parent::__construct($balance);
        $this->overdraftLimit = $overdraftLimit;
    }
    public function withdraw($amount) {
        if ($this->balance - $amount >= $this->overdraftLimit) {
            parent::withdraw($amount);
        } else {
            echo "Dépassement de découvert autorisé\n";
        }
    }
}
// Exemple d'utilisation
```

```
$savingsAccount = new SavingsAccount(1000);
$checkingAccount = new CheckingAccount(500);
$savingsAccount->deposit(200);
$checkingAccount->withdraw(600);
echo "Solde du compte d'épargne: " . $savingsAccount->getBalance() . "\n";
echo "Solde du compte courant: " . $checkingAccount->getBalance() . "\n";
$savingsAccount->applyInterest();
echo "Solde du compte d'épargne après l'application des intérêts: " .
$savingsAccount->getBalance() . "\n";
```

Dans cet exemple :

- Nous avons une interface BankAccount qui définit les méthodes communes pour tous les types de comptes bancaires.
- La classe BaseAccount implémente cette interface et encapsule le solde du compte ainsi que les opérations de dépôt et de retrait.
- Les classes SavingsAccount et CheckingAccount héritent de BaseAccount et spécialisent son comportement. Le compte d'épargne applique un taux d'intérêt, tandis que le compte courant gère un découvert autorisé.
- L'encapsulation est réalisée en protégeant les propriétés et en fournissant des méthodes publiques pour interagir avec elles.

Question 1: Qu'est-ce qu'une interface en programmation orientée objet ?

Réponse: Une interface en programmation orientée objet est une structure qui définit un ensemble de méthodes qu'une classe doit implémenter. Elle spécifie le comportement d'un objet sans fournir l'implémentation concrète de ces méthodes. En PHP, une interface est définie à l'aide du mot-clé interface.

Question 2: Quel est le rôle de l'encapsulation dans la programmation orientée objet ?

Réponse: L'encapsulation est un principe de programmation orientée objet qui consiste à regrouper les données (variables) et les méthodes (fonctions) qui les manipulent au sein d'une même entité, appelée objet. Cela permet de cacher les détails internes de l'objet et de limiter l'accès direct aux données, favorisant ainsi la modularité et la réutilisabilité du code.

Question 3: Expliquez le concept d'héritage en programmation orientée objet.

Réponse: L'héritage est un mécanisme en programmation orientée objet qui permet à une classe (appelée classe dérivée ou sous-classe) de hériter des attributs et des méthodes d'une autre classe (appelée classe de base ou superclasse). La classe dérivée peut étendre ou spécialiser le comportement de la classe de base en ajoutant de nouvelles méthodes ou en redéfinissant les méthodes existantes. Cela favorise la réutilisabilité du code et permet de créer des hiérarchies de classes.

Question 4: Comment pouvez-vous utiliser l'héritage en PHP ?

Réponse: En PHP, vous pouvez utiliser l'héritage en déclarant une classe qui étend une autre classe à l'aide du mot-clé extends. Par exemple, class MaClasse extends ClasseDeBase {...}. La classe MaClasse héritera alors de toutes les propriétés et méthodes de ClasseDeBase, et vous pourrez les utiliser dans MaClasse comme si elles lui appartenaient directement.