Création d'un Système de Gestion de Compte Bancaire

Objectif générale :

Développer un système de gestion de comptes bancaires en Python comprenant deux classes principales : CompteBancaire et Banque. La classe CompteBancaire encapsulera les détails d'un compte bancaire individuel, tandis que la classe Banque gérera un ensemble de comptes bancaires.

Partie 1 :

Objectif

Développer une classe **CompteBancaire** en Python qui encapsule les détails d'un compte bancaire et fournit des méthodes pour effectuer des opérations bancaires de base.

1. **Définition de la Classe CompteBancaire**
   - Créez une classe **CompteBancaire** avec un constructeur. La classe doit avoir des attributs privés pour le numéro de compte, le nom du titulaire et le solde.
   - Le constructeur doit initialiser ces attributs. Le solde doit être optionnel avec une valeur par défaut de 0.
2. **Méthodes pour les Opérations Bancaires**
   - Ajoutez une méthode **depot(montant)** pour déposer de l'argent sur le compte.
   - Ajoutez une méthode **retrait(montant)** pour retirer de l'argent du compte. Assurez-vous de vérifier si le solde est suffisant pour le retrait.
   - Ajoutez une méthode **afficher_solde()** pour afficher le solde actuel du compte.
3. **Encapsulation**
   - Assurez-vous que tous les attributs de la classe sont privés. Utilisez des getters et des setters si nécessaire pour accéder ou modifier ces attributs de manière contrôlée.
   - Par exemple, vous pourriez vouloir empêcher la modification directe du solde en dehors des méthodes de dépôt et de retrait.
4. **Test du Code**
   - Créez une instance de **CompteBancaire** et testez toutes les méthodes pour vous assurer qu'elles fonctionnent correctement.
   - Essayez de déposer un montant, de le retirer, d'afficher le solde, et vérifiez la gestion des erreurs (comme le retrait d'un montant supérieur au solde).

Partie 2 :

Objectif

Développez une classe Banque pour gérer un ensemble de **CompteBancaire**.
La classe Banque doit permettre d'ajouter et de supprimer des comptes, d'effectuer des transactions, d'afficher les informations d'un compte spécifique, et de lister tous les comptes.

1. **Définition de la Classe Banque**
    - La classe **Banque** doit contenir une liste ou un dictionnaire pour stocker les instances de **CompteBancaire**.
    - Elle devrait fournir des méthodes pour ajouter un nouveau compte, supprimer un compte, et effectuer des transactions entre les comptes.
2. **Méthodes de la Classe Banque**
    - **Ajouter un Compte**: Une méthode pour ajouter un nouveau **CompteBancaire** à la banque.
    - **Supprimer un Compte**: Une méthode pour supprimer un compte existant de la banque.
    - **Effectuer Transaction**: Une méthode pour effectuer des transactions entre deux comptes (par exemple, transférer de l'argent d'un compte à un autre).
    - **Afficher les Comptes**: Une méthode pour afficher les détails de tous les comptes dans la banque.
3. **Gestion des Comptes**
    - Chaque fois qu'un nouveau compte est créé ou supprimé, la liste ou le dictionnaire des comptes doit être mis à jour en conséquence.
    - Assurez-vous que les transactions sont valides (par exemple, le compte source a suffisamment de fonds).
4. **Test du Code**
    - Créez une instance de **Banque** et testez les méthodes pour ajouter, supprimer, et effectuer des transactions entre les comptes.
    - Vérifiez que les opérations sont exécutées correctement et que les soldes des comptes sont mis à jour après chaque transaction.

Partie 3 :

Menu Interactif

Créez un menu interactif pour permettre aux utilisateurs d'effectuer les actions suivantes :

1. Créer un nouveau compte.
2. Supprimer un compte existant.
3. Effectuer un dépôt sur un compte.

4. Effectuer un retrait d'un compte.
5. Afficher le solde d'un compte.
6. Lister tous les comptes dans la banque.
7. Quitter le programme.

Scénario de Test

oici un scénario de test détaillé pour évaluer votre système de gestion de comptes bancaires en Python, en utilisant les classes **CompteBancaire** et **Banque** ainsi que le menu interactif :

Scénario de Test

1. Initialisation de la Banque
   - Lancer le programme.
   - Le menu interactif s'affiche.
2. Création de Comptes
   - Sélectionner l'option pour créer un nouveau compte.
   - Entrer les informations nécessaires (numéro de compte, nom du titulaire).
   - Répéter l'opération pour créer plusieurs comptes avec différents numéros et titulaires.
3. Dépôt d'Argent
   - Sélectionner l'option pour effectuer un dépôt.
   - Choisir un compte existant.
   - Saisir un montant à déposer.
   - Vérifier que le solde du compte a été mis à jour correctement.
4. Retrait d'Argent
   - Sélectionner l'option pour effectuer un retrait.
   - Choisir le même compte ou un autre.
   - Saisir un montant à retirer.
   - Vérifier que le solde du compte a été mis à jour correctement.
   - Tenter un retrait avec un montant supérieur au solde pour tester la gestion des erreurs.
5. Affichage du Solde
   - Sélectionner l'option pour afficher le solde.
   - Choisir un compte.
   - Vérifier que le solde affiché est correct.
6. Liste des Comptes
   - Sélectionner l'option pour lister tous les comptes.
   - Vérifier que tous les comptes créés sont listés avec leurs soldes corrects.
7. Suppression de Compte
   - Sélectionner l'option pour supprimer un compte.
   - Choisir un compte existant.
   - Vérifier que le compte a été supprimé et n'apparaît plus dans la liste des comptes.
8. Quitter le Programme
   - Sélectionner l'option pour quitter le programme.
   - Vérifier que le programme se ferme correctement.

Générateur de Page HTML avec Classes pour Éléments

Objectif

Développer un système en Python pour créer une page HTML. Utilisez des classes distinctes pour gérer différents éléments HTML (tels que **H1**, **Img**, **P**, **A**, et **Div**), en mettant en pratique les concepts de classes, de constructeurs et d'encapsulation.

1. **Création de Classes pour Balises HTML**
    - Définissez des classes séparées pour chaque type de balise HTML : H1, Img, P, A, et Div.
    - Chaque classe doit avoir un constructeur pour initialiser ses attributs, qui peuvent inclure du texte, des sources d'images, des hyperliens, etc., ainsi que des attributs HTML optionnels.
2. **Encapsulation et Gestion des Attributs**
    - Assurez-vous que les attributs de chaque balise sont bien encapsulés. Permettez la modification de ces attributs uniquement à travers des méthodes publiques si nécessaire.
3. **Méthode de Rendu HTML**
    - Chaque classe doit implémenter une méthode __str__ pour générer et retourner son propre code HTML.
4. **Assemblage de la Page HTML**
    - Utilisez une classe PageHTML pour assembler les différents éléments HTML. Cette classe devrait permettre d'ajouter des éléments à la page et de générer le code HTML final.
5. **Test de Fonctionnalité**
    - Testez votre système en créant une instance de PageHTML et en y ajoutant différents éléments HTML. Générez ensuite le code HTML et affichez le résultat pour vérification.

Exigences Techniques

- Les classes pour les balises HTML doivent gérer leurs propres attributs et contenir la logique nécessaire pour produire le code HTML.
- La classe PageHTML doit être capable de gérer une collection d'éléments HTML et de produire le code HTML complet de la page.
- Le système doit permettre la flexibilité dans l'ajout d'attributs HTML aux éléments.

Exemple d'Utilisation

Créez une page HTML contenant un titre H1, un paragraphe P, une image Img, un lien hypertexte A, et un conteneur Div. Testez différentes configurations d'attributs pour chaque élément et observez le rendu final du code HTML généré.

Générateur de Page HTML Simple sans Héritage

Objectif

Concevoir un système en Python pour créer et générer une page HTML simple, en utilisant des classes pour représenter différents éléments HTML, tout en évitant l'utilisation de l'héritage.

1. **Définition de la Classe Élément HTML**
    - Créez une classe **ElementHTML**. Cette classe aura des attributs pour le nom de la balise, le contenu et les attributs HTML (comme les classes, id, etc.).
    - Le constructeur doit initialiser ces attributs.
    - Utilisez une méthode **__str__** pour générer le code HTML de cet élément sous forme de chaîne de caractères.
2. **Encapsulation des Attributs**
    - Utilisez l'encapsulation pour protéger les attributs de l'élément HTML. Les attributs de la balise, comme le nom, le contenu et les attributs HTML, doivent être privés.
    - Fournissez des méthodes pour obtenir et définir ces attributs de manière sécurisée.
3. **Gestion des Attributs HTML**
    - Ajoutez une méthode pour ajouter ou modifier les attributs HTML de l'élément (comme **ajouter_attribut** ou **definir_attribut**).
4. **Classe pour la Page HTML**
    - Créez une classe **PageHTML** qui contiendra une liste d'éléments HTML.
    - Ajoutez une méthode pour ajouter des éléments à la page.
    - Ajoutez une méthode **generer_html** qui construit et retourne le code HTML complet de la page.
5. **Test du Générateur de Page HTML**
    - Créez une instance de **PageHTML**.
    - Ajoutez des éléments HTML en utilisant des instances de **ElementHTML** pour différents types de contenu (par exemple, un titre, un paragraphe, une image).
    - Générez le HTML de la page et affichez-le.