Année Scolaire
2023-2024

RAPPORT PROJET (POO)

Formation et Niveau : Master 2 data analyst
Professeur : M. David ALBERT
Etudiant : Khalid OURO-ADOYI

Projet POO : Logiciel de Gestion de Comptes Bancaires en Python

DESCRIPTION :

Ce projet a pour objectif de mettre en place un système de gestion de comptes bancaires en utilisant le langage de programmation Python. L'objectif principal est de modéliser les relations entre différentes entités ou classes impliquées dans le domaine bancaire. Les principales entités comprennent les opérations bancaires, les comptes bancaires, les clients, les institutions bancaires, et une base de données pour stocker ces informations.

Le système permet de créer des opérations bancaires, des comptes bancaires de différents types tels que l'épargne et le courant, des clients associés à des adresses spécifiques, et des institutions bancaires. Les clients sont également dotés d'attributs tels que le salaire, les dépenses mensuelles, et peuvent calculer le montant du prêt ainsi que le taux d'intérêt applicable.

La structure du code intègre une classe de base pour les comptes bancaires avec des classes dérivées pour les comptes d'épargne et courants. Les relations entre les clients, les comptes bancaires et les institutions bancaires sont modélisées à travers des méthodes d'ajout et de retrait. De plus, une base de données factice est utilisée pour stocker les informations sur les institutions et les clients.

Ce projet démontre une approche orientée objet pour la modélisation du domaine bancaire, en permettant la création, la gestion et la relation entre les différentes entités impliquées dans un système bancaire simplifié.

LES CLASSES

1. **OperationBancaire**: Représente une opération bancaire avec des attributs tels que le type d'opération, le montant et la date. Cette classe est indépendante et associée aux comptes bancaires.

2. **Client**: Représente un client de la banque avec des attributs tels que le nom, le prénom, l'adresse, etc. Les clients peuvent posséder plusieurs comptes bancaires, modélisés par une relation d'association. Ils ont également des attributs spécifiques tels que le salaire et les dépenses mensuelles, permettant le calcul du montant du prêt et du taux d'intérêt.

3. **CompteBancaire**: Classe de base pour les comptes bancaires, avec des attributs comme le numéro de compte, le type de compte, le client associé, et une liste d'opérations. Les comptes bancaires peuvent être de différents types, tels que compte d'épargne ou compte courant. Les opérations bancaires (dépôts, retraits, etc.) sont modélisées comme une relation d'agrégation ou de composition.

4. **CompteEpargne**: Hérite de CompteBancaire et représente un compte d'épargne avec des attributs spécifiques comme le solde, le taux d'intérêt et la date d'ouverture. Cette sous-classe illustre une relation d'héritage.
5. **CompteCourant**: Hérite de CompteBancaire et représente un compte courant avec des attributs spécifiques comme le solde, le découvert autorisé et les frais de gestion. Cette sous-classe illustre également une relation d'héritage.

6. **InstitutionBancaire**: Représente une institution bancaire avec des attributs tels que le nom et une liste de clients. Les clients peuvent appartenir à une institution bancaire, illustrant une relation d'association.
7. **Database** : permet de stocker les informations sur les institutions et les clients créés dans le système. Elle dispose de méthodes pour ajouter des institutions et des clients à la base de données. Cette classe joue le rôle de conteneur global pour gérer les entités du système bancaire.

Chacune de ces classes dispose de méthodes spécifiques pour gérer les opérations bancaires, les clients, les comptes et les institutions bancaires. L'exemple d'utilisation fourni à la fin du code démontre comment créer une institution bancaire, un client, des comptes bancaires et des opérations, tout en respectant les conditions liées à l'adresse du client.

LES DIFFERENTES RELATIONS ENTRE NOS CLASSES

LES DIFFÉRENTES RELATIONS

1. Relation d'héritage :

CompteEpargne et CompteCourant sont des sous-classes de CompteBancaire : cela signifie que CompteEpargne et CompteCourant héritent des attributs et des méthodes de la classe CompteBancaire. Cette relation d'héritage permet une hiérarchie de classes, où les sous-classes peuvent étendre ou spécialiser les fonctionnalités de la classe parente.

2. Relation d'association :

Il existe une relation d'association entre InstitutionBancaire et Client car un client appartient à une institution bancaire. Cette relation souligne la connexion entre ces deux entités, montrant que les clients sont affiliés à une institution bancaire particulière.

3. Relation d'agrégation :

CompteBancaire, CompteEpargne, CompteCourant et la classe OperationBancaire ont une liste d'opérations bancaires. Cette relation d'agrégation ou de composition indique que ces classes contiennent des objets de la classe OperationBancaire. Les opérations bancaires sont associées aux comptes bancaires, montrant ainsi la dépendance entre ces entités.

4. Méthode Super() :

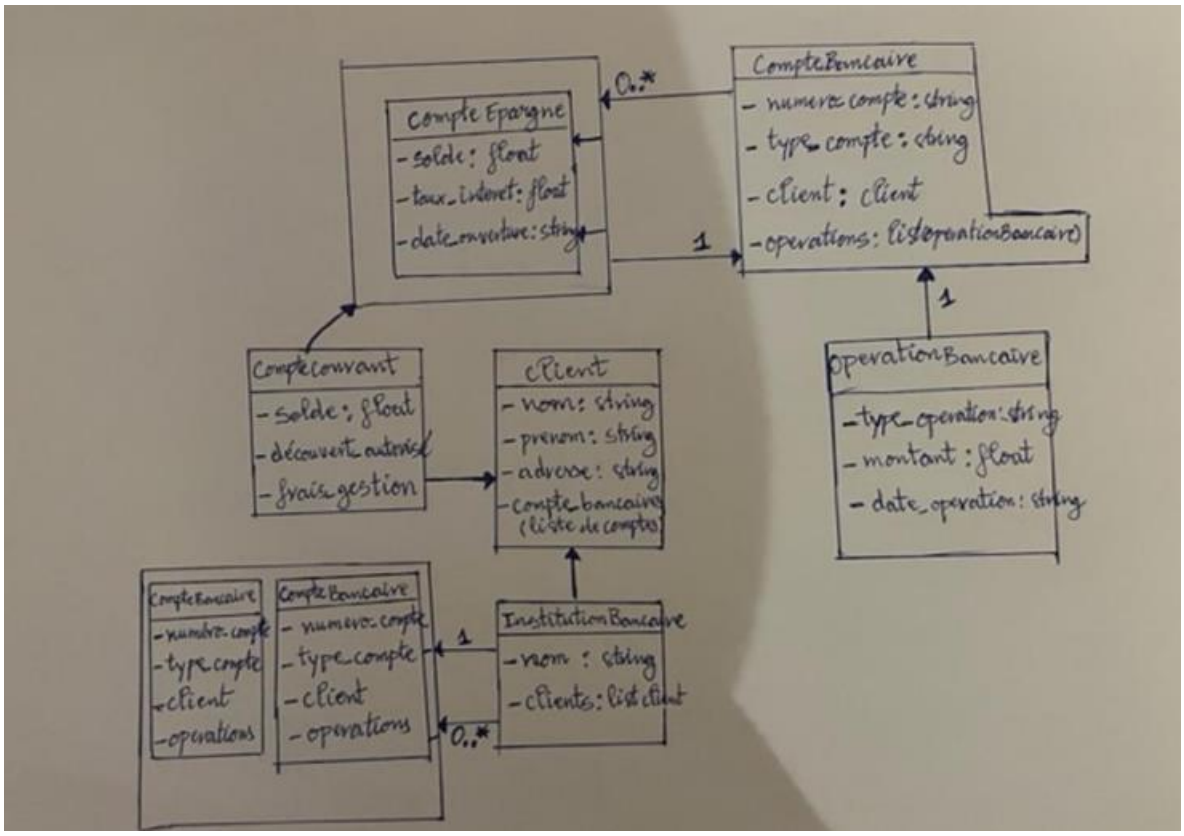
La méthode super() est présente dans les sous-classes CompteEpargne et CompteCourant, qui héritent de la classe CompteBancaire. Super() assure que les attributs communs sont initialisés correctement avant d'ajouter les attributs spécifiques aux sous-classes. C'est un moyen de réutiliser le code de la classe parente tout en personnalisant les fonctionnalités dans les sous-classes.

DIAGRAMME UML

Dans ce diagramme simplifié, les flèches représentent les relations d'association entre les différentes classes. On observe les relations d'héritage entre CompteEpargne et CompteBancaire, ainsi qu'entre CompteCourant et CompteBancaire. Les relations d'agrégation ou de composition sont également indiquées entre CompteBancaire et OperationBancaire, ainsi qu'entre Client et CompteBancaire. Enfin, il y a une relation d'association entre Client et InstitutionBancaire.

Sur ce diagramme UML, les associations sont représentées par des lignes avec des flèches indiquant la direction de l'association. Les multiplicateurs (comme "0..", "1") sont ajoutés pour indiquer les cardinalités. Par exemple, "0.." signifie "zéro ou plusieurs", et "1" signifie "exactement un".

Dans un diagramme UML, le signe "-" devant un attribut ou une opération indique que cet élément est privé, c'est-à-dire qu'il n'est pas accessible directement depuis l'extérieur de la classe qui le contient. Cela représente l'encapsulation, un principe clé de la programmation orientée objet.



Fonctionnalités Principales :

1. Exploration de l'onglet "Comptes" :

   Affichage de la liste de tous les comptes disponibles.

2. Utilisation de l'onglet "Opérations" :

   Ajout de dépôts, retraits et autres transactions (calcule de montant de prêt, taux d'intérêt).

Utilisation de la Base :

1. Création d'un Nouveau Compte :

   Cliquez sur "Nouveau Compte" et suivez les étapes pour renseigner les détails requis.

2. Ajout d'une Opération :

   - Sélectionnez un compte.
   - Cliquez sur "Nouvelle Opération".

Difficulté :

- Erreurs Inattendues lors de l'Ajout d'une Opération :

- Solutions suggérées :
    - Amélioration de la gestion des erreurs.
    - Fourniture de messages détaillés pour guider l'utilisateur.

Perspectives d'Améliorations :

1. Intégration de Fonctionnalités Avancées :
    - Gestion de budgets pour une meilleure maîtrise financière.
2. Synchronisation avec les Services Bancaires en Ligne :
    - Ajout d'une fonction de synchronisation pour faciliter les mises à jour automatiques.

Ces perspectives d'améliorations visent à enrichir les fonctionnalités de l'application, à fournir une meilleure expérience utilisateur et à optimiser la performance du code. La gestion des erreurs devrait être renforcée pour offrir des retours clairs en cas de problèmes.

RESUME

Fonctionnalités du logiciel:

- Création de clients, de comptes bancaires de différents types, et d'institutions bancaires selon la domiciliation.
- Attribution de comptes bancaires à des clients.
- Réalisation d'opérations bancaires (dépôts salaires, réalisation des depenses, emprunt et calcule de taux d'intérêt) associées à des comptes bancaires.
- Gestion de clients appartenant à une institution bancaire.
- Possibilité d'ajouter, retirer des comptes bancaires, et d'ajuster les informations des clients.

Avantages :

- Utilisation de la programmation orientée objet pour modéliser les entités et leurs relations.
- Structure modulaire permettant une extension facile des fonctionnalités.
- Encapsulation pour assurer la sécurité des données en limitant l'accès direct aux détails internes des classes.

Utilisation :

Un exemple d'utilisation est fourni à la fin du code pour illustrer la manière dont les différentes classes peuvent être instanciées et utilisées pour créer un système de gestion de comptes bancaires fonctionnel.