Projet Compte Bancaire

Table des matières

Projet Compte Bancaire 1

Objectifs 2

Evolution du produit 2

Note 2

Etape 1 3

Spécifications : 3

Aide : 3

Etape 2 4

Spécifications 4

Etape 3 5

Spécifications 5

Etape 4 6

Spécifications 6

Etape 5 7

Spécifications 7

Etape 6 8

Objectifs

- Travailler la notion d'héritage
- Lire un diagramme en UML pour la représentation des classes
- Utiliser la surcharge des méthodes avec et sans paramètres optionnels
- Ecrire soi-même les classes de tests

Evolution du produit

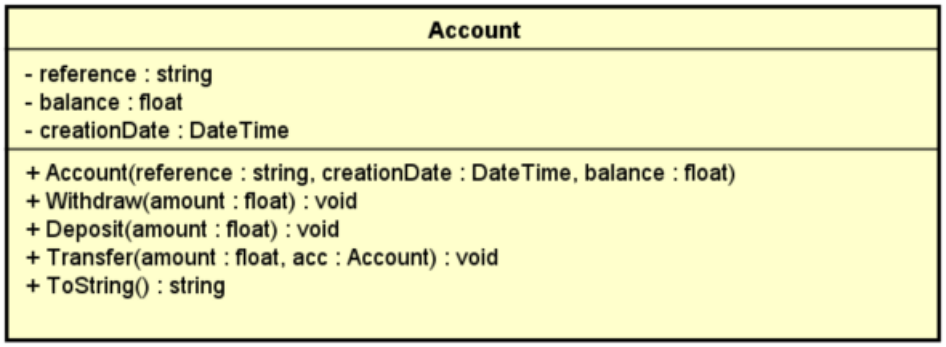
Nous allons travailler de manière itérative. Il s'agit, étape après étape, d'améliorer le produit sans déstabiliser les fonctionnalités acquises dans les étapes précédentes (non régression).

Note

Pour des raisons de simplification dans la lecture du diagramme de classes, les accesseurs ne sont pas représentés dans le diagramme de classes, à vous de déterminer s'ils sont nécessaires en fonction des besoins du projet.
Dans ce projet, les attributs de la classe sont en général alimentés par le constructeur.

Etape 1

Nous allons effectuer une première version d'une application de type « **Library** » qui implémentera la classe représentée dans ce diagramme de classes UML (ce schéma s'enrichira au fur et à mesure du projet) :



Spécifications

- Le constructeur accepte de créer un compte même si le paramètre « balance » n'est pas passé en paramètre lors de l'instanciation (paramètres optionnels)
- Les attributs privés « reference » et « balance » sont disponibles via des accesseurs en lecture, seul l'attribut Balance est modifiable depuis une autre classe via un accesseur en écriture.
- Les méthodes sont toutes implémentées
    - Déposer de l'argent (deposit) met à jour automatiquement le solde du compte
    - Retirer de l'argent (withdraw) met à jour automatiquement le solde du compte
- Si le montant souhaitant être retiré est supérieur au solde du compte, l'exception AmountTooHighException est levée.
- Tous les attributs et méthodes sont documentés.
- Créer les tests automatiques pour les méthodes Withdraw, Deposit, Transfer et ToString. La méthode ToString() donnera une description du compte (valeur du n° de compte, du solde et de la date de création par exemple).

Aide :

```
public void SomeMethod(int a, int b = 0)
{
   //some code
}
```

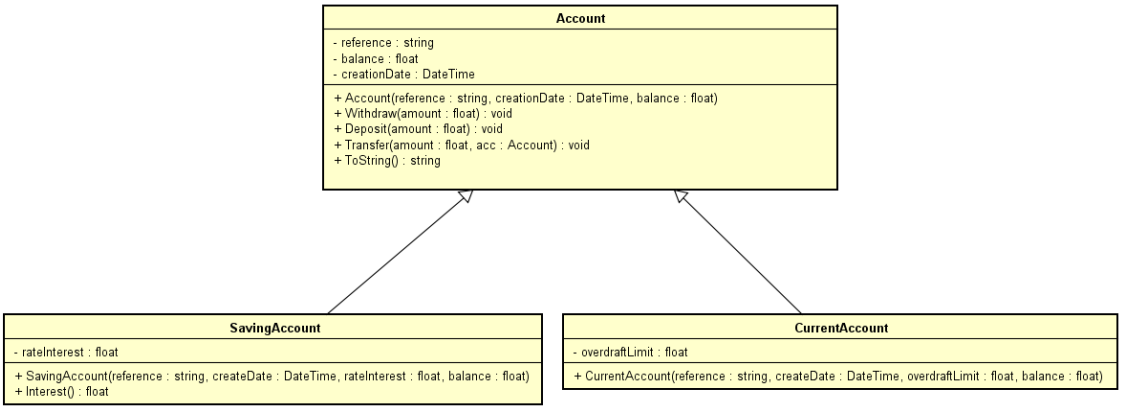
Dans cette exemple, soit le paramètre b est passé en paramètre, soit il recevra par défaut la valeur 0.

Etape 2

Maintenant que nous avons une classe « Account » testée et fonctionnelle, nous désirons ajouter la possibilité :

- De générer des intérêts pour les comptes épargnes
- D'un découvert pour les comptes courants.

Nous allons utiliser la possibilité, offerte par la POO, de partir de la classe compte et de l'enrichir fonctionnellement dans la classe « SavingAccount » et « CurrentAccount ». Ceci en s'appuyant sur le mécanisme d'héritage.
Voici en UML la représentation de ce que nous désirons obtenir :



Spécifications

- La classe « SavingAccount » hérite des attributs et fonctionnalités de « Account ». De même pour la classe « CurrentAccount ».
- La méthode « Interest » calcule et retourne l'intérêt produit par le solde du compte (au moment où l'on appelle la méthode)
    - Si le solde du compte est positif, l'intérêt est positif
    - Si le solde du compte est à zéro, l'intérêt est zéro
    - Si le solde du compte est négatif, l'intérêt sera également négatif

  Tester cette méthode à l'aide de tests automatiques.
- Tous les attributs et méthodes sont documentés.

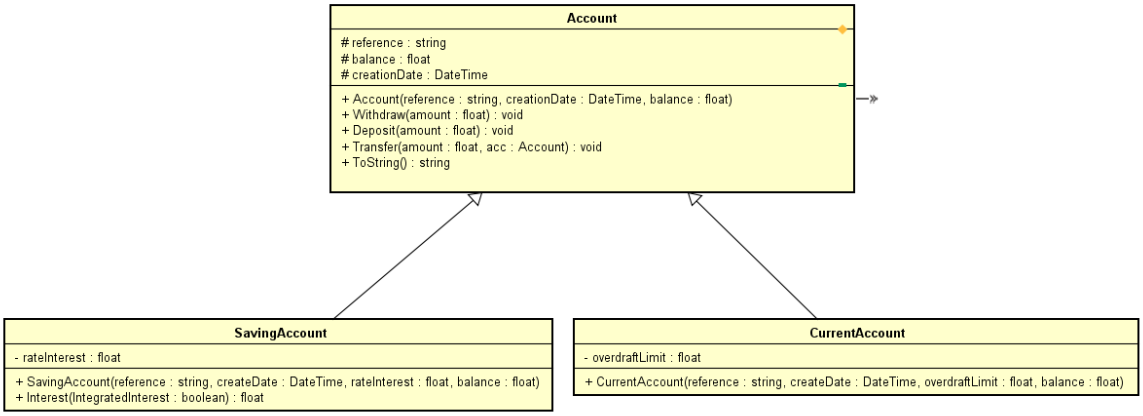
Etape 3

Nous allons encore faire évoluer le « SavingAccount ». Pour l'instant cette classe est capable de calculer l'intérêt, mais ne l'ajoute pas au solde du compte.

Nous aimerions enrichir la méthode « Interest » d'un paramètre qui nous permet, soit :

- de calculer et d'intégrer les intérêts au compte
- de ne faire que calculer les intérêts

Voici le digramme UML représentant la nouvelle situation désirée :

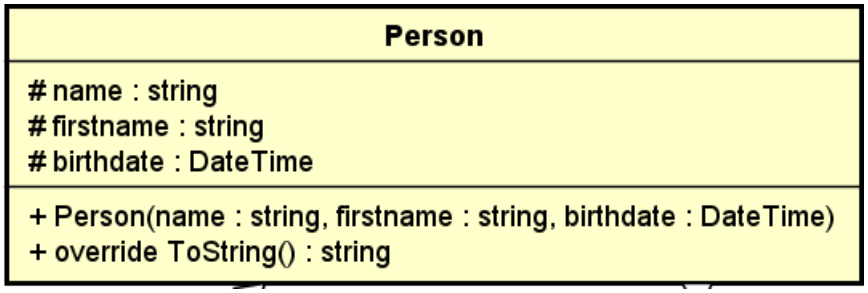


Spécifications

- La classe « SavingAccount » permet d'intégrer les intérêts dans le solde du compte sans utiliser d'accesseur en écriture, mais bien en impactant directement l'attribut « balance » hérité de la classe « Account »

Etape 4

Afin de pouvoir intégrer une notion de client et d'employés dans notre application, nous allons ajouter une classe supplémentaire selon le diagramme de classe UML suivant :

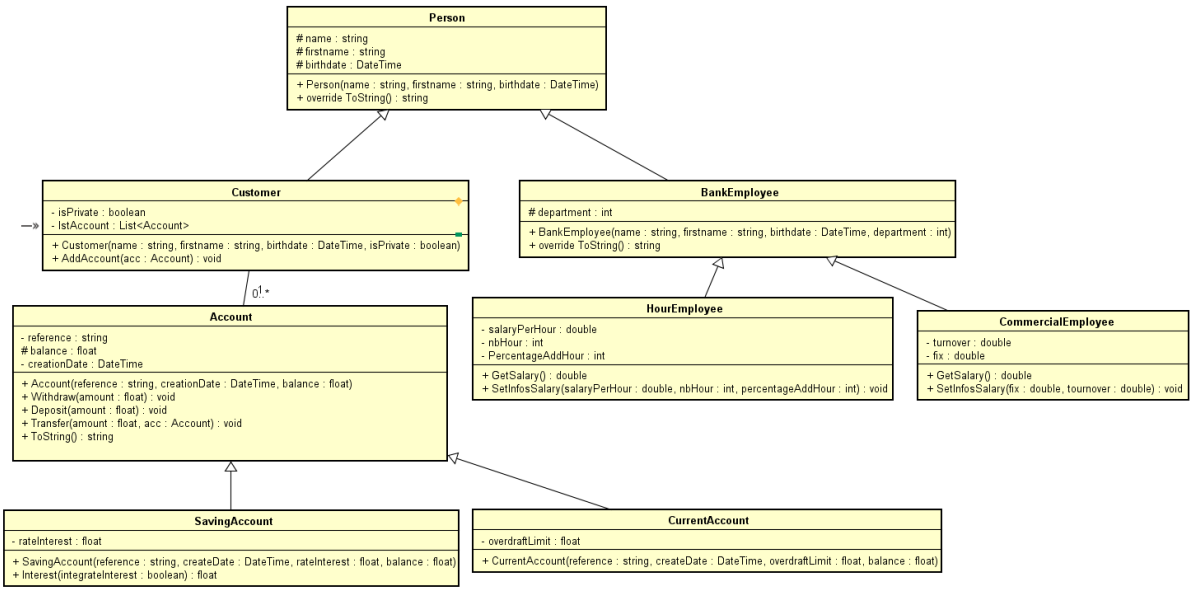


Spécifications

- La méthode ToString() permet d'afficher la description de la personne (nom, prénom et date de naissance).

Etape 5

A l'étape précédente nous avons créé une classe qui nous permet d'intégrer une notion d'employé de banque et de client.
Etudiez le diagramme de classes ci-dessous et modifier votre code en fonction des spécifications ci-dessous.



Spécifications

- La classe « BankEmployee » hérite de la classe « Person »
- La classe « Customer » hérite de la classe « Person »
- Le classe « HourEmployee » hérite de la classe « BankEmployee » et représente les employés de la banque qui sont payés selon le nombre d'heures effectuées dans la semaine. Ils sont payés à un certain tarif horaire et leurs heures supplémentaires (au-delà de la charge horaire due de 40h) sont payées 30 % ou 50% de plus que les heures normales.
- La classe « CommercialEmployee » hérite de la classe « BankEmployee » et représente les commerciaux qui, eux, sont payés avec une somme fixe à laquelle on ajoute 1 % du chiffre d'affaires qu'ils ont fait dans la semaine.
- La méthode GetSalary() retourne le salaire de l'employé par semaine.
- La méthode SetInfosSalary() permet de mettre à jour les informations sur le salaire de l'employé. Le paramètre nbHour représente le nombre d'heures total effectuées dans la semaine.
- « HourEmployee » et « CommercialEmployee » contiennent un constructeur chacun avec les paramètres nécessaires (à préciser par vos soins). Ils ne sont pas représentés dans le diagramme de classe.
- Créer les méthodes de tests automatiques pour les méthodes SetInfosSalary() et GetSalary().

Etape 6

Créer un diagramme de séquences présentant la création de 2 comptes, un compte courant et un compte épargne, pour un nouveau client.

Dans la même solution, ajouter un projet de type Console. Dans ce projet, faire référence au projet de type « Library ». Implémenter le diagramme de séquences présenté ci-dessus. Afficher également la description des deux comptes.

Si vous avez du temps ….

Créer également un projet de type « Windows Form ». Dans ce projet, faire référence au projet de type « Library ».

Créer un formulaire permettant d'entrer les informations sur un client et compte. Utiliser les méthodes créées dans le projet de type « Library ».