Secteur Tertiaire Informatique

Filière « Etude et développement »

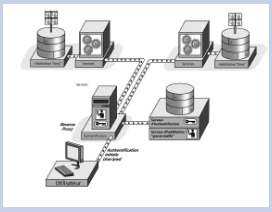
Gestion de comptes bancaires

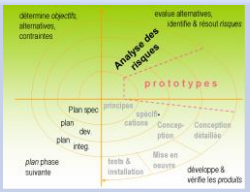
Programmation Orientée Objet

Apprentissage

Mise en situation

Evaluation





1. INTRODUCTION

L'objectif de ce TP est de concevoir un programme en console basé sur une approche objet et permettant de gérer les comptes en banque de client.

Vous allez créer deux classes :

- « **Account** » : représente des comptes bancaires
- « **Customer** » : représente des clients d'une banque

Avec ces classes vous pourrez implémenter les fonctionnalités suivantes :

- Créer des comptes bancaires (avec vérification de leur IBAN)
- Créer des clients
- Associer des comptes bancaires aux clients

Le code attendu doit suivre les règles syntaxiques et les conventions de nommage du langage Java

Pour vous aider dans le développement une base de code est disponible à l'adresse suivante : **https://github.com/afpa-learning/poo-account**

2. IMPLEMENTATION DES CLASSES

2.1 IMPLEMENTATION DE LA CLASSE « ACCOUNT »

Dans un premier temps, il vous faudra créer une classe représentant un compte bancaire.

Un compte bancaire sera définit par les caractéristiques suivantes :

- **IBAN :** Internation Bank Account Number, code unique international d'un compte, de type chaîne de caractères ;
- **« balance » (montant) :** argent sur le compte (peut être négatif) , entier
- **« overdraftAuthorization » :** découvert autorisé, entier

Dans ce projet, nous considérerons que les montants des comptes sont en valeur entières.

Cette première version de la classe employée peut être graphiquement représentée en utilisant le langage UML comme présenté ci-dessous:

Account
- iban: String - balance: int - overdraftAuthorization: int
+ constructeur + getters + setters + toString(): + addMoney(amount: int): int + removeMoney(amount: int): int + transfer(otherAccount: Account, amount: int): void

Quelques détails sur les méthodes à implémenter :

- « addMoney » : permet d'ajouter un montant sur le compte
- « removeMoney » : permet de retirer un montant du compte (si possible)
- « transfer » : prend en paramètre un **deuxième** objet de la classe « Account » et de transférer un certain montant
-

Attention

Si un retrait d'argent dépasse l'autorisation de découvert l'opération ne devra pas être possible !

Un message devra être affiché en console.

A faire

Complétez la classe « Account » pour y ajouter tous les attributs ainsi que les « **getters** » (appelés accesseurs en français) et les « **setters** » (appelés mutateurs) et implémentez les méthodes.

A faire

Ajoutez la méthode « **toString()** » qui aura pour objectif de transformer un objet en une représentation sous forme de chaîne de caractères.

Pour plus d'information concernant la méthode « **toString()** » :
https://codegym.cc/fr/groups/posts/fr.986.mthode-java-tostring

A faire

Instanciez quelques objets de la classe « Account » avec des informations différentes et essayez toutes les fonctionnalités.

2.1.1 Vérification de l'IBAN

A faire

Ajoutez, dans la classe « Account » une méthode de vérification de l'IBAN nommée « **checkIBAN** ».

Une fois que cette méthode est fonctionnelle, utilisée la pour empêcher l'instanciation d'un compte ou la modification de l'IBAN si celui-ci n'est pas bon.

Plus d'information sur l'algorithme de vérification d'IBAN disponible ici : https://fr.wikipedia.org/wiki/International_Bank_Account_Number

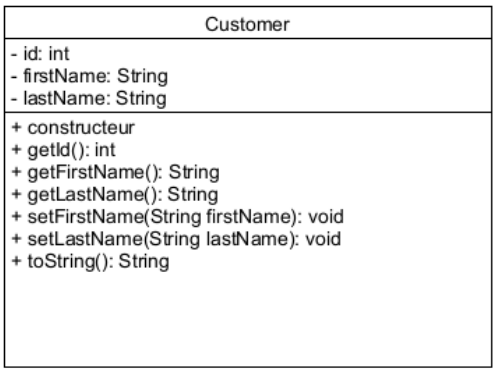
2.2 IMPLEMENTATION DE LA CLASSE « CUSTOMER »

2.2.1 Première version

Vous allez créer une seconde classe qui sera en relation avec « Account », la classe « Customer ».

Un « Customer » est défini par les caractéristiques suivantes :
- Id : identifiant unique, entier
- « **firstName** » : prénom, string
- « **lastName** » : nom, string

Première représentation UML :

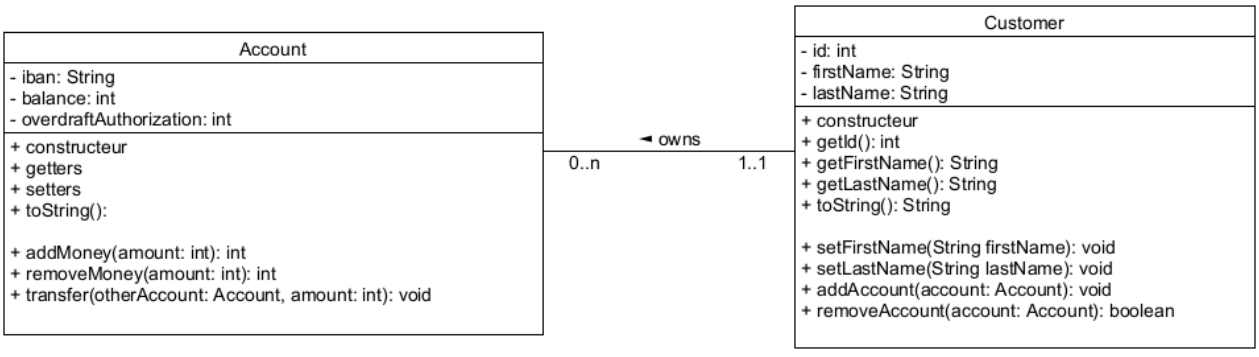


A faire

Implémentez une première version de cette classe, comme présenté par le diagramme UML.

2.2.2 Relation entre « Customer » et « Account »

Vous allez maintenant implémentez la relation entre un « Customer » et un compte bancaire, comme présenté par la figure suivante :



Règle métier indiquée sur ce diagramme : un client peut avoir un ou plusieurs comptes bancaires mais peut aussi ne pas en avoir.

En UML, cette règle est indiquée par le lien « **owns** » qu'il y a entre «la classe « Customer » et « **Account** » et les **cardinalités** de chacun des liens :

- **0..n** : indique que le client peut avoir de **0 à un nombre indéterminé** de comptes bancaire
- **1..1 :** indique qu'un compte bancaire ne **peut avoir qu'un seul client**

Pour implémenter une telle contrainte vous allez pouvoir ajouter un **attribut** de type « **ArrayList<Account>** » dans la classe « **Customer** », par exemple avec le code suivant :

```
private ArrayList<Account> accounts = new ArrayList<Account>();
```

Vous pouvez maintenant ajouter les méthodes d'ajout et de suppression des comptes :

- « **addAccount(Account account)** »
- « **removeAccount(Account account)** »

Ces méthodes devront faire appel à des méthodes de la classe « **ArrayList** ».

Pour plus d'informations sur la classe « **ArrayList** » veuillez-vous référer au site suivant : https://www.w3schools.com/java/java_arraylist.asp

A faire

Une fois la « **ArrayList** » ajoutée et les méthodes implémentées testez votre code à partir de la fonction « main » en instanciant plusieurs objets des classes « Account » et « Customer » et en ajoutant/supprimant des comptes à un client en utilisant les méthodes « **addAccount » et « removeAccount ».**

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)
Michel Coulard – Formateur Evry
Chantal Perrachon – IF Neuilly sur Marne

Date de mise à jour : 21/06/2024