TP n° 1
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

1 Représentation d'un système bancaire (simplifié)

Pour cet exercice, portant sur le domaine bancaire, chaque classe développée possédera une méthode `toString()`. Sa réalisation comporte plusieurs étapes :

1. Créer les classes `CompteBancaire` et `Client` conformément à la documentation fournie sur Moodle [1] veillant au principe d'encapsulation et en vous aidant de la classe `TestCompteBancaire` fournie.
2. Puis définir une nouvelle classe `CompteBancaireRémunéré` qui héritera de la classe `CompteBancaire`. Cette classe possédera un attribut représentant le taux de rémunération et un attribut de classe indiquant un taux d'intérêt par défaut (par exemple 3 %) à utiliser lorsque celui-ci n'est pas précisé lors de la création du compte. Prévoir trois constructeurs :
   - un constructeur permettant d'initialiser le solde sans préciser le taux d'intérêt, la valeur prise sera alors la valeur précisée par l'attribut de classe ;
   - un constructeur permettant d'initialiser le solde et le taux d'intérêt ;
   - un constructeur initialisant le solde à zéro et le taux d'intérêt à la valeur par défaut fixée arbitrairement à 3 %.
3. Définir une méthode `crediterInteretMensuel()` permettant de créditer le compte des intérêts mensuels générés par la valeur du solde (pour rendre la méthode plus simple, mais inexacte, elle créditera le compte du douzième des intérêts annuels).
4. Tester votre implémentation de la classe `CompteBancaireRémunéré` avec la classe `TestCompteBancaireRémunéré`.
5. Créer deux nouvelles classes `Particulier` et `Entreprise` héritant de `Client` en définissant un attribut spécifique à chacune de ces sous-classes (ex. prénom pour un particulier, numéro de SIRET pour une entreprise).
6. Concernant la classe `CompteBancaire`, pour que le numéro de compte soit attribué automatiquement (sans devoir être fourni en paramètre), vous utiliserez une variable de classe, qui sera incrémentée à chaque création d'un compte et dont la valeur servira à initialiser le numéro de compte.
7. Puis créer une classe `Banque` permettant de gérer à la fois les clients et leurs comptes (ajout et suppression).

   Pour éviter de devoir associer systématiquement les clients à la banque ainsi que les comptes créés, la classe `Banque` offrira des méthodes qui prendront en charge la création des clients et des comptes ainsi que leur association à la banque (pour les noms de méthodes se reporter à la classe `TestBanque` fournie).

1. Consulter la documentation en ouvrant le fichier `index.html` en premier.

La figure 1 ci-dessous montre les différentes classes ainsi que leurs relations.

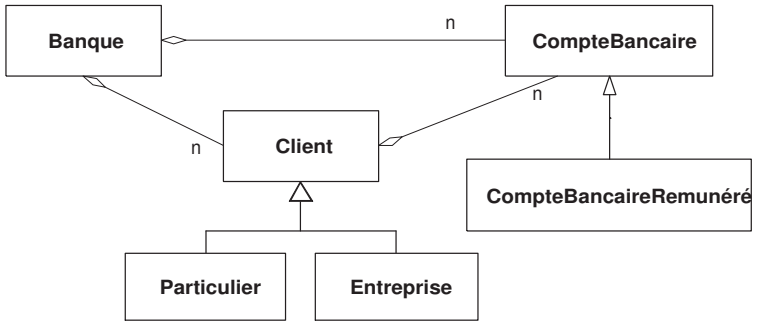

FIGURE 1 – Les différentes classes du système bancaire

Les associations débutant par un « losange » sont des relations dites d'agrégation et signifient « peut contenir » ou « peut posséder », la cardinalité s'exprimant à l'extrémité de l'association (ici `n`). Ainsi on peut déduire de ce diagramme qu'une banque peut posséder plusieurs (`n`) clients et plusieurs comptes et qu'un client peut posséder plusieurs comptes.

La cardinalité est importante pour déterminer le type de structure de données à utiliser. Ainsi une cardinalité `n` se traduira par la présence d'une liste (ou d'un tableau) dans la classe « portant » le losange. Le type de la liste correspondra à la classe située à l'autre extrémité de l'association. Par exemple, on peut déduire de ce diagramme que la classe `Banque` possédera une liste de type `Client`.

8. Compléter la classe `TestBanque` fournie pour vérifier les contraintes suivantes :
   — un client ne peut pas ouvrir plus de trois comptes bancaires au sein de la même banque ;
   — un client peut être supprimé que s'il n'a aucun compte associé ;
   — un transfert entre deux comptes appartenant à deux établissements bancaires différents sera facturé 5 euros au débiteur (cette somme devra donc être débitée en plus du montant à transférer). Il faudra ici tester la valeur des deux comptes après l'opération.

   Les tests seront toujours sur le même modèle : on effectue un appel à une méthode puis on compare le résultat obtenu à celui attendu. Si les deux diffèrent alors un message d'erreur sera affiché.