NFA032 : TP n°8 (Références)

10 avril 2018

Objectifs et rappels

L'objectif de ce Tp est de travailler sur une application qui exploite la posibilité d'avoir accès à un même objet via différentes variables ou *partage de données*. Prenons l'exemple d'un objet Compte de Java : il est représenté par son adresse mémoire, dite *référence* (ou pointeur). Si cette référence est contenue dans plusieurs variables (par exemple, plusieurs cases de tableau), on aura accès à l'objet en passant par chacune de ces variables. Des cases de tableaux ou des variables dans d'autres objets peuvent pointer vers ce compte et donc modifier son contenu. La figure 1 montre le dessin où un tableau et plusieurs variables pointent vers des objets partagés. Etudiez ce code et dessin en vous assurant de bien comprendre le pourquoi des flèches représentées.

Première partie : Banque, Comptes et Titulaires

Pour réaliser cet exercice recopiez dans votre projet le paquetage `sourcesExo1`.

On veut travailler sur la modélisation d'une banque et de titulaires de compte détenus par cette banque. Une banque possède une liste de comptes. Un titulaire possède un nom et tous les comptes dont il est titulaire. On voit donc qu'un même objet compte sera à la fois dans la liste de la banque, et dans celle de son titulaire.

Nous vous donnons du code à compléter pour les classes Banque, Titulaire, ainsi qu'une classe Compte. Quelques brèves explications sur ces classes suivent. **Attention : il pourra être utile ou nécessaire d'ajouter d'autres méthodes dans ces classes**.

Titulaire

Un titulaire possède un nom et une variable `ArrayList<Compte>` permettant de stocker ses comptes. Il peut également ajouter un nouveau compte dans sa liste.

Banque

Les comptes dans la Banque ont tous un numéro unique et deux comptes différents n'auront pas le même numéro. La Banque devra implanter une méthode méthode pour obtenir le bilan de soldes de tous ses comptes, et une méthode pour tester si un numéro correspond à un compte de la banque.

Seule la banque peut créer des nouveaux comptes. Quand elle crée un nouveau compte, elle l'ajoute à la liste des comptes de son titulaire et elle l'ajoute aussi à sa propre liste de comptes. De même, chaque nouveau compte créé se voit attribuer par la banque un nouveau numéro, exclusif pour ce compte. Pour

```
Java
    }
    public double getSolde() {
        return solde;
    }
    public int getNum() {
        return numero;
    }
    public void depot(double n){
      solde = solde+n;
    }
    public void retrait(double m) {
      solde = solde-m;
    }
public static void main(String[] args) {
    Compte [] t = new Compte[4];
    Compte c0 = new Compte(1,10);
    Compte c1 = new Compte(2,50);
    t[0]= c0; t[2]= t[0];
    t[1]= c1;
}
```

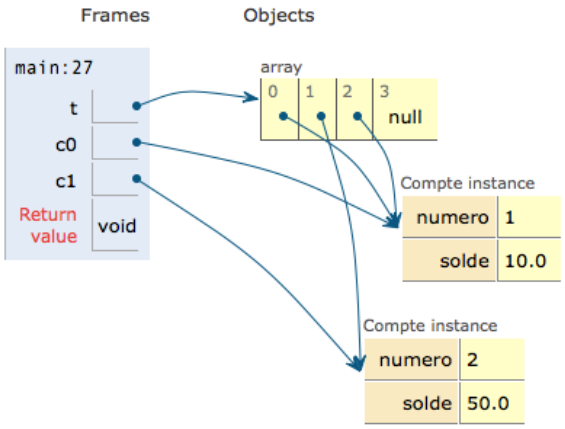

FIGURE 1 – un dessin avec références

réaliser la création de comptes, il faut une méthode dans Banque qui crée un compte avec en paramètre le titulaire, et qui lui attribue un numéro de compte (unique).

Question 1 : étude du code fourni + dessin

Le code fourni est un est incomplet et donc ne fonctionne pas correctement. Le but de cette première question est de comprendre ce code avant de le modifier ou de l'exécuter.

Complétez la méthode `main` de la classe `TestOpsBanque` qui réalise une situation où trois titulaires, Paul, Pierre et Fatima ont des comptes dans une banque *BNP*. Paul a deux comptes, les autres n'en ont qu'un.

Dessinez la situation en faisant apparaître sur un schéma tous les objets et tableaux en vous inspirant du dessin de la figure 1.

Question 2 : compléter le code fourni

Complétez le code des classes données en respectant les commentaires mis dans le code. Il peut être utile d'ajouter de nouvelles méthodes dans ces classes.

Question 3 : méthodes getCompte

Écrivez deux méthodes `getCompte`, l'une dans la classe `Banque` l'autre dans la classe `Titulaire`, qui renvoient un compte dont le numéro a été donné en paramètre. S'il n'y a pas de compte avec ce numéro, la méthode doit lever une exception.

Question 4 : virements

On désire écrire deux méthodes de virement. La première effectue un virement depuis un compte dont on est titulaire vers un compte dont on ne connaît que le numéro (et dont on n'est pas forcément titulaire), et dont on connaît la banque qui possède ce compte. La deuxième méthode est une version surchargée de la première qui effectue un virement entre deux comptes d'un même titulaire. Version surchargée signifie qu'elle doit porter le même nom que la première méthode mais qu'elle doit avoir des paramètres différents.

Dans quelle classe faut-il placer ces méthodes ?

Question 5 : main (suite)

Ajoutez au `main` de `TestOpsBanque` le code permettant d'appeler les deux méthodes de virement.

Partie 2 : égalité d'objets

Nous travaillerons ici sur l'égalité entre valeurs de type références : égalité entre objets et égalité entre tableaux. Il existe deux moyens de comparer les objets et tableaux en Java :

1. **L'opérateur == compare les adresses** : si deux variables tableaux ou objet *contiennent la même adresse mémoire* alors ce test d'égalité renvoie `true`. Sinon, ce test renvoie `false`. En d'autres termes, on teste si les deux variables pointent vers (partagent) le même espace mémoire.
2. La méthode `boolean equals(Object o)` est pré-définie par défaut sur tout objet et sur tout tableau. Son comportement diffère selon le cas :
   — **sur les types objet prédéfinis** comme String, Integer : compare *compare les contenus des objets testés et non pas leur adresses*. Par exemple, si nous avons :

   ```
   Integer n1 = new Integer(1);
   Integer n2 = new Integer(1);
   ```

   les tests `n1.equals(n2)` et `n2.equals(n1)` renvoient true, alors que `n1==n2` renvoie false.
   — **sur les types objet non pré-définis et sur les tableaux** : compare les adresses et donne donc les mêmes résultats que ==. Cette méthode peut être rédéfinie dans toute classe pour se comporter d'une autre manière, mais il faut respecter certaines règles pour qu'elle s'accorde bien au fonctionnement des bibliothèques java. On vous renvoie à la documentation java ou au cours NFA035 pour les détails.

Sans avoir à redéfinir `equals` il est possible d'écrire des méthodes qui comparent deux objets en comparant les contenus de leurs variables d'instance respectives.

Question 1

Exécutez la méthode main de la classe `testEgalite`. Etudiez son code et pour chaque ligne où un commentaire vous pose l'indique, insérez une explication courte sur ce qui est affiché par cette ligne de code.

Question 2

Ajouter du code à la méthode `main` de la classe `TestOpsBanque` pour montrer par un calcul et un affichage que le compte de Pierre est le même objet que le compte qui a ce numéro dans la classe `Banque`.

Question 3

Ajouter à la classe `Banque` une méthode `authentifierCompte` permettant d'authentifier un compte. Cette méthode a pour but de s'assurer qu'un objet `Compte` passé en paramètre est bien un des comptes gérés par la banque. Le résultat sera un booléen : `true` si le compte est bien un compte de la banque, `false` sinon. Il faut faire ici un test d'identité : l'objet passé en paramètre doit avoir la même adresse qu'un des comptes de la banque.

Question 4

On va améliorer la méthode d'authentification pour détecter les tentatives d'escroquerie basées sur la création de comptes qui usurpent le numéro d'autres comptes.

Pour ce faire, on va ajouter une méthode `equals` dans la classe `Compte` qui va uniquement comparer les numéros de comptes : si deux objets ont le même numéro, `equals` renvoie `true` et sinon, elle renvoie `false`.

Ensuite, modifier la méthode `authentifierCompte` pour qu'elle lève une exception `AlerteEscroquerieExcep` si l'objet passé en paramètre n'est pas un compte de la banque mais qu'il est `equals` à un des comptes de la banque.

Question 5

Modifiez le code de la classe `TestOpsBanque` pour réaliser une exécution dans laquelle l'exception `AlerteEscroquerieException` est levée.

A rendre

La totalité des exercices de ce Tp (sauf les dessins) sont à rendre en séance, au format zip.