TP Java : Une mini-banque interactive

Objectifs du TP : Manipuler des objets et les faire interagir entre eux

---

Dans ce TP, nous allons modéliser le fonctionnement d'une banque. Cela nous permettra de faire interagir ensemble, plusieurs types d'objet différents : le compte, le client, la banque.

Vous écrirez chaque classe dans un fichier différent. Vous écrirez également un programme principal qui possédera uniquement la méthode main, dans laquelle vous testerez chacune de vos classes au fur et à mesure de leur écriture.

Exercice 1 : Diagramme de classe (UML)

Question 1.1: Réaliser un diagramme

Avant de commencer, nous allons réaliser un diagramme de classe du TP. Sachez qu'il est nécessaire de lire le sujet plusieurs fois avant de commencer le diagramme.
Vous pourrez utiliser l'outil de votre choix, comme https://www.draw.io/

Exercice 2 : Le compte

Question 2.1: La classe Compte

La première étape est d'écrire une classe Compte.
Celle-ci doit contenir :

- 2 attributs privés : son numéro et son solde
- 4 méthodes publiques :

```
/* pour faire un depot sur le compte. */
void depot(float valeur){}

/* pour faire un retrait sur le compte. */
void retrait(float valeur){}

/* pour obtenir la valeur du solde */
float getSolde(){}

/* pour afficher le solde */
void afficherSolde(){}
```

Question 2.2: Pour faire un virement

Jusque-là, les fonctions et les méthodes que nous utilisions avaient pour arguments des types primitifs, c'est-à-dire des int, des float...Comme la méthode `depot(float valeur)` par exemple.

Maintenant, nous allons voir qu'il est possible d'utiliser un objet comme argument d'une méthode : Ajoutons une méthode à la classe compte qui aura pour effet de faire un virement vers un autre compte.

Cette méthode aura 2 arguments : la somme à déplacer et le compte destinataire.

```
/* pour virer de l'argent d'un compte vers un autre compte */
void virer(float valeur, Compte destinataire){}
```

Exercice 3 : Le client

Question 3.1: La classe Client

Jusque-là, les attributs des objets que nous avons étudiés étaient des types primitifs (int, float…).
Il est également possible d'utiliser un objet comme attribut d'un autre objet.
Étant donné qu'un client possède un compte, chaque objet client doit avoir un objet compte comme attribut.

Nous pourrions donc déclarer un compte dans la classe Client, de la même manière qu'un attribut primitif :

```
public class Client {
    private Compte compte = new Compte();
}
```

Écrire la classe Client. Celle-ci doit posséder :

- 2 attributs privés : son nom et son compte
- 3 méthodes :
    - `getNom` (renvoie le nom du client)
    - `getSolde` (renvoie le solde du compte unique du client
    - `afficherSolde` (affiche le solde du client)

Question 3.2: Le nom du client

Chaque client possède un attribut nom et une méthode pour le renvoyer. Lors de sa création, Il n'est pas possible de choisir le nom du client. Pour arranger, nous allons définir un constructeur avec des arguments.

Par défaut, toutes les classes possèdent un constructeur sans argument :

```
public class Exemple {
}
```

Si nous ne le voyons pas, il est présent comme la déclaration équivalente ci-dessous :

```
public class Exemple {

    public Exemple() {
    }
}
```

Les constructeurs que nous avons vu précédemment ne possédaient pas d'arguments. Nous allons donc en créer un comme ci-dessous :

```
    public Client(String nomDuClient) {
            …
    }
```

Grâce ce constructeur (qu'il faudra compléter), on peut préciser le nom du client à la création :

```
public class Main {

    public static void main(String[] args) {
        Client nouveauClient =new Client("Boris");
    }
}
```

Dans la classe Client, Créer un constructeur comme indiqué ci-dessus.

Question 3.3: Plusieurs comptes pour un client

Dans notre mini-banque interactive, un client peut avoir plusieurs comptes. Pour cela, au lieu d'avoir un attribut compte de type Compte, nous allons utiliser un attribut comptes, qui sera un tableau contenant plusieurs comptes.

En java, les tableaux peuvent contenir des types primitifs (int, float…), mais également des types Objet.

```
/* Exemple d'un tableau d'entiers d'une taille fixe de 10 valeurs */
int[] entiers = new int[10];

/* Exemple d'un tableau d'utilisateurs d'une taille fixe de 10 valeurs */
Utilisateur[] utilisateurs = new Utilisateur[10];
```

Modifiez la classe Client selon cette nouvelle organisation :

- Un Client pourra posséder 100 comptes

Attention, lorsque votre client sera créé (grâce à son constructeur), un compte par devra être créé aussi, dans le tableau.

```
/* Exemple de création d'un Compte dans le tableau */
Compte[0] comptes = new Compte();
```

En plus de l'attribut *comptes*, vous pouvez créer :

- L'attribut `nbComptes` (contient le nombre de compte du client)
- La méthode `ajouterCompte` (ajoute un nouveau compte)

Vous devez également modifier la méthode `getSolde` pour qu'elle renvoie dorénavant la somme du solde de tous les comptes du client.

Exercice 4 : La banque

Question 4.1: La classe banque

Comme avec le client qui peut contenir plusieurs comptes, une banque peut contenir plusieurs clients.

Écrire la classe Banque. Celle-ci doit posséder :

- 2 attributs privés : son nom et son compte
- 3 méthodes :
  - `ajouterClient` (avec le nom du client en paramètre)
  - `bilanClient` (affiche le bilan de tous les comptes du client)
  - `afficherBilan` (affiche un bilan général de tous les compte de tous les clients)

Exercice 5 : La banque interactive

Question 5.1: une classe à part entière

Comme le compte, le client, la banque, nous pouvons mettre en place une classe *BanqueInteractive* pour gérer les interactions de notre mini-banque.

Ajouter à cette classe une méthode *interaction()* qui entame un dialogue avec l'utilisateur pour faire fonctionner la banque.

Voici un exemple de dialogue tel qu'il doit apparaître :

La saisie clavier de l'utilisateur est précédée par le signe –

```
Quelle opération voulez-vous effectuer ?
1) Ajouter un client
2) Effectuer une opération sur un client
3) Afficher un bilan général

- 1

Entrez le nom du client :

- M. Dupont

Le client M. Dupont a et crée.
Quelle opération voulez-vous effectuer ?
1) Ajouter un client
2) Effectuer une opération sur un client
3) Afficher un bilan général

- 2

Quel client ?

1) M. Dupont

- 1

Quelle opération voulez-vous effectuer sur le client M. Dupont ?

1) Afficher un bilan
2) Faire un retrait
3) Faire un dépôt
4) Faire un virement

...
```

Pour mieux structurer votre programme, il vous est conseillé de créer plusieurs méthodes intermédiaires pour répondre à cette question.

Par exemple, pour demander à l'utilisateur le nom du nouveau client et pour l'ajouter :

```
public void interactionAjoutClient(){
}
```

Autre exemple, pour demander à l'utilisateur de choisir un client, puis d'initier une interaction concernant le client sélectionné (bilan, retrait, dépôt, virement) :

```
public void interactionOperationClient(){
}
```

Cette interaction, pourra être effectuée par une méthode d'interaction qui sera insérée dans Client. Pour simplifier, supposons que les dépôts et les retraits se font sur le 1er compte de chaque client, et que les virements s'effectuent entre 2 comptes d'un même client.

Exercice 6 : Renflouement des comptes

Question 6.1: solde négatif et renflouement

Chaque client possède un compte que l'on peut considérer comme son compte courant.

Si son solde devient négatif, la banque peut renflouer ce compte en effectuant des virements depuis les autres comptes du client.

Ajoutez à la classe Client une méthode `renflouer()` qui effectue, si c'est nécessaire et si c'est possible, ce renflouement.

Ajoutez à la classe Banque une méthode `renflouer()` qui effectue le renflouement pour tous les clients.

Adapter le menu de la méthode `interaction()` pour pouvoir effectuer ce renflouement à la demande.