TP Java n°2 : Une mini-banque interactive

IUP I.I.E.S 1
13 Février
J. Carme

Objectifs du TP : Manipuler des objets plus complexes, les faire interagir entre eux

---

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

Comme d'habitude, vous écrirez chaque classe dans un fichier différent. Vous écrirez également un programme principal, ne possédant que la méthode *main*, dans laquelle vous testerez chacune de vos classes au fûr et à mesure de leur écriture.

Exercice 1: Le compte

Question 1.1: La classe Compte

La première étape est d'écrire une classe *Compte*. Celle-ci doit contenir deux attributs privés: son numéro et son solde, ainsi que quatre méthodes publiques:

```
void depot(float valeur); /* pour faire un dépot sur le compte. */
void retrait(float valeur); /* pour faire un retrait sur le compte. */
float getSolde(); /* pour obtenir la valeur du solde */
void afficherSolde(); /* pour afficher le solde */
```

Question 1.2: 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*... C'est le cas par exemple de la méthode *depot* que nous avons écrite à la question précédente. Nous allons maintenant voir qu'il est possible d'utiliser un objet comme argument d'une méthode.

Nous allons ajouter une méthode à la classe compte qui aura pour effet de faire un virement vers un autre compte. Cette méthode aura donc deux arguments: la somme à déplacer, et le compte destinataire:

```
void virer(float valeur, Compte destinataire)
```

Exercice 2: Le client

Question 2.1: La classe Client

Jusque là, les attributs des objets que nous avons étudiés était des types primitifs (*int*, *float*...). Il est également possible d'utiliser un objet comme attribut d'un autre objet.

Ici, chaque client possède un compte. Chaque objet client doit donc avoir un objet compte comme attribut. Il suffit de le déclarer dans la classe au même titre que n'importe quel autre attribut:

```
private Compte compte = new Compte();
```

Vous devez maintenant écrire la classe Client. La classe client doit posséder deux attributs privés: son nom et son compte. Elle doit également posséder trois méthodes: $getNom$, qui renvoie le nom du client, $getSolde$, qui renvoie le solde total du client (c'est-à-dire le solde de son unique compte), $afficherSolde$, qui affiche le solde du client.

Question 2.2: Le nom du client

Comme vous l'avez remarqué, chaque client possède un attribut *nom*, une méthode pour renvoyer le nom du client, mais aucun moyen de choisir le nom du client. Pour arranger, nous allons utiliser le constructeur, d'une manière un peu différente de celle que l'on a vu jusque là.

Les constructeurs que nous avons vu précédemment de possédait pas d'arguments. Mais il est également possible de mettre un argument au constructeur, pour paramétrer l'objet que l'on construit. Par exemple, le constructeur de la classe Client pourrait ressembler à cela:

```
public Client(String nom_du_client) {
   ...
}
```

Dans ce cas, on choisit le nom du client à la création de l'objet, en procédant ainsi:

```
Client nouveau_client=new Client(‘‘Alexandre’’);
```

Créez un constructeur comme indiqué ci-dessus qui attribut un nom à chaque classe créée.

Question 2.3: Plusieurs comptes pour un client

Nous allons maintenant permettre à chaque client d'avoir plusieurs comptes. Pour cela, au lieu d'avoir un attribut *compte*, nous allons avoir un attribut *comptes*, qui sera un tableau contenant plusieurs comptes.

Vous avez déjà précédemment étudié les tableaux. Nous allons maintenant voir qu'il est possible de faire des tableaux d'objets.

Par exemple, pour déclarer l'attribut *comptes*, on peut écrire:

```
public Compte [] comptes=new Compte[100];
```

Notez que la valeur 100 correpond à la taille du tableau, c'est à dire au nombre maximum de comptes que chaque client peut posséder. Attention! En créant le tableau, vous n'avez encore créé aucun compte, juste le tableau les contenant. Pour créer le compte correpondant à la première case du tableau, par exemple, il faut procéder ainsi:

```
comptes[1]=new Compte();
```

Il vous faudra faire cela pour chaque nouveau compte créé.

Modifiez la classe Client selon cette nouvelle organisation. Vous devez ajouter, en plus de l'attribut *comptes*, l'attribut *nbComptes* qui contient le nombre de compte du client, une méthode $ajouterCompte$ qui ajoute un nouveau compte. Vous devez également modifier la méthode $getSolde$ pour qu'elle renvoie la somme du solde de tous les comptes du client.

Exercice 3: La banque

Question 3.1: La classe banque

De même qu'un client peut contenir plusieurs comptes, une banque peut contenir plusieurs clients. Crééez une classe $BanqueInteractive$, possédant entre autre les méthodes $ajouterClient(stringnom_du_client)$, $bilanClient(intnumero_du_client)$ qui affiche le bilan de tous les comptes d'un client, et $afficherBilan()$ qui affiche un bilan général de tous les compte de tous les clients.

Question 3.2: Un fonctionnement interactif

Ajouter à la classe *Banque* une méthode *interaction*() qui entame un dialogue avec l'utilisateur pour faire fonctionner la banque. Voici un example de dialogue tel qu'il doit apparaître. Les réponses de l'utilisateur sont précédées par le signe -.

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

- 1

Entrez le nom du client:

- M. Dupont

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

- 2

Quel client?

1) M. Dupont

- 1

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

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

...
```

Pour mieux structurer votre programme, il vous est conseiller de créer plusieurs méthodes "intermédiaires" pour répondre à cette question. Par exemple, une méthode *interactionAjoutClient*() pourra s'occuper de demander le nom du nouveau client à l'utilisateur et de l'ajouter et une méthode *interactionOperationClient*() pourra s'occuper des opérations sur un client, et appeler si besoin est d'autres méthode plus spécifiques.