TD 1 : Rappels de programmation orientée objet

Objectifs pédagogiques :

- **notion de classe**
- **encapsulation**
- **diagramme de classes UML**
- **agrégation/composition d'objets**

Introduction

Dans ce premier TD, nous allons concevoir et programmer en Java une application simple dans un domaine métier connu de tous : la gestion de comptes bancaires. Lors d'une phase préliminaire d'analyse du domaine métier, des concepteurs ont élaboré le diagramme de classes UML suivant :

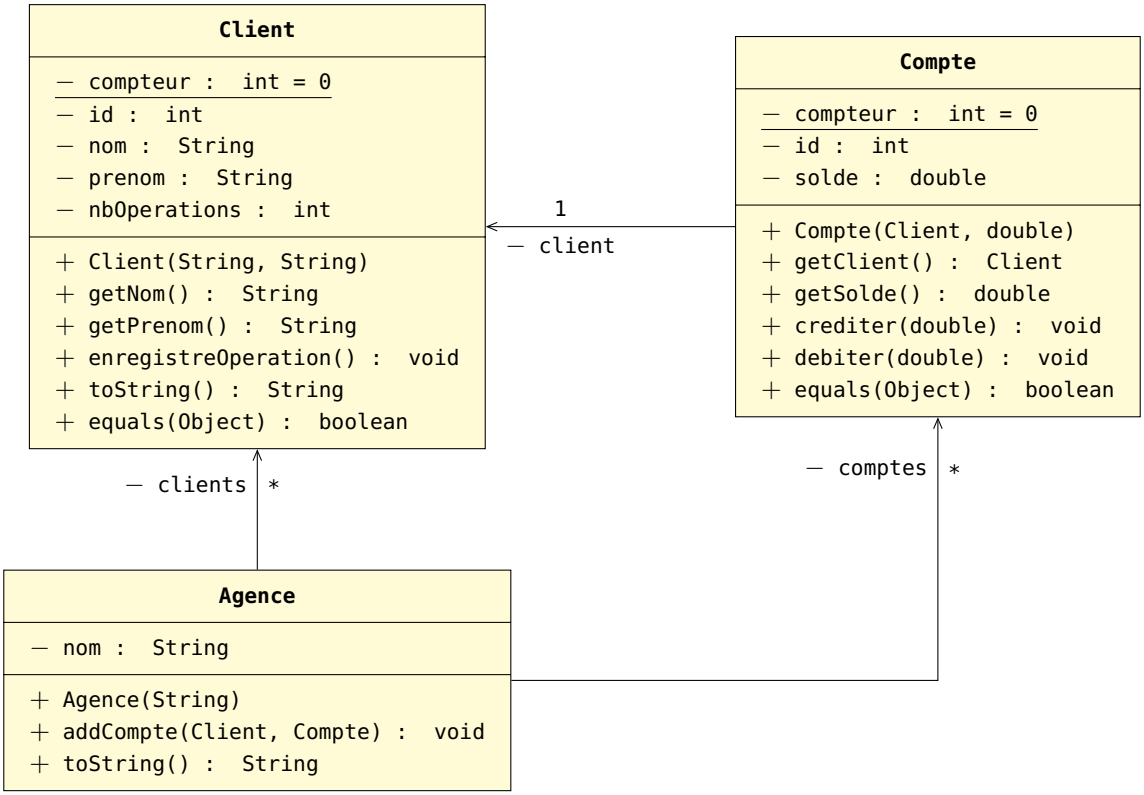

Figure 1: Diagramme UML pour les classes Agence, Compte et Client.

Un résumé des notations UML utilisées en cours et TD est proposé en annexe, à la fin de ce document.

Les associations représentent des attributs de classes. Par exemple :

- `comptes` est un attribut privé (identifié par le symbole −) de type `List<Compte>` ou `Compte[]` (car de multiplicité *) appartenant à la classe `Agence` ;
- `client` est un attribut privé de type `Client` (de multiplicité 1) de la classe `Compte`.

Attention à ne pas représenter un attribut *à la fois* sous forme d'association et dans le compartiment du milieu de la classe (il faut choisir).

Pour être sûr de ne pas entrer en conflit avec d'autres applications bancaires, nous utilisons le nom de paquetage `pobj.banque` pour notre implémentation.

Nous donnons ci-dessous l'implémentation de la classe `Client` :

Client.java

```java
package pobj.banque;

public class Client {
  private static int compteur = 0;
  private int id;
  private final String nom;
  private final String prenom;
  private int nbOperations;

  /**
  * Construit un client de nom et prénom spécifiés
  * @param nom le nom du client
  * @param prenom le prénom du client
  */
  public Client(String nom, String prenom) {
    id = compteur++;
    this.nom = nom;
    this.prenom = prenom;
    nbOperations = 0;
  }

  /**
  * Accède au nom du client
  * @return le nom du client
  */
  public String getNom() { return nom; }

  /**
  * Accède au prénom du client
  * @return le prénom du client
  */
  public String getPrenom() { return prenom; }

  /**
  * Enregistre une nouvelle opération effectuée par le client
  */
  public void enregistreOperation() { nbOperations++; }

  /** Méthode standard : représentation textuelle */
  @Override public String toString() {
     return "Client(nom=" + nom + ", prénom=" + prenom + ", nbOps=" + nbOperations + ")";
  }

  /** Méthode standard : comparaison de clients */
  @Override public boolean equals(Object o) {
    if (o == this)
      return true;
    if (!(o instanceof Client))
      return false;
    Client c = (Client)o;
    return (c.nom.equals(nom) && c.prenom.equals(prenom));
  }
}
```

1.1 Structure de la classe et vocabulaire

Question 1. Quel est le nom de la classe (relatif/absolu) ?
Question 2. Quel est le nom du type correspondant ?
Question 3. Quel est son paquetage ?
Question 4. Quel est le nom et où doit se trouver le fichier source de la classe ?
Question 5. Identifiez les différentes parties de la classe, notamment :

- les attributs ;
- les constructeurs ;
- les méthodes d'accès (*getter*, *setter*) ;
- les méthodes standard ;
- les méthodes de traitement.

Question 6. Expliquez les mots-clés utilisés dans le code et donnez une justification.
Question 7. Comment sont calculés les identifiants de chaque client ? Expliquez le principe de ce calcul.

1.2 Classe `Compte`

La classe `Compte` garde une référence sur un client, et elle contient le solde du compte.
Question 8. Proposez une implémentation Java conforme à ce qui apparaît dans le diagramme de classes. La méthode `boolean equals(Object o)` renvoie vrai si deux comptes ont le même identifiant. Concernant l'attribution de l'identifiant, utilisez le même procédé que pour la classe `Client`. Les concepteurs demandent de comptabiliser les opérations sur le compte : vous devrez donc incrémenter le compteur d'opérations du client à chaque crédit et à chaque débit de son compte.

1.3 Classe `Agence`

L'agence bancaire gère directement le fichier clients et la base des comptes. Pour stocker les clients et les comptes dans cette classe nous avons besoin d'utiliser un objet permettant de sauvegarder une collection d'objets. Nous avons à notre disposition les tableaux Java et les `ArrayList`, dont une description succincte est donnée en annexe de ce TD.
Question 9. Quelle(s) différence(s) y a-t-il entre les tableaux Java et les `ArrayList` ? Comparez les syntaxes d'instanciation, les instructions pour y lire/écrire des données et l'accès à la taille du tableau.
Question 10. Proposez une implémentation de la classe `Agence` basée sur des `ArrayList` pour gérer les comptes et les clients de l'agence. L'implémentation ne doit pas stocker de doublon, c'est-à-dire qu'un client et un compte ne doivent être référencés qu'au plus une fois dans l'agence.
Question 11. Donnez les affichages issus des instructions suivantes.

BanqueTest.java

```
package pobj.banque.test;

import pobj.banque.Agence;  // ou alors, d'un seul coup: import pobj.banque.*;
import pobj.banque.Client;
import pobj.banque.Compte;

public class BanqueTest
{
   public static void main(String[] args)
   {
      Agence agence = new Agence("Banque Impopulaire");
      Client client = new Client("Dupont", "Jean");
```

```
      System.out.println("Nom client : " + client.getNom());
      System.out.println("Prénom client : " + client.getPrenom());

      Compte compte1 = new Compte(client, 1000);
      Compte compte2 = new Compte(client, 2000);
      agence.addCompte(client, compte1);
      agence.addCompte(client, compte2);
      System.out.println(compte1.getSolde());
      System.out.println(compte2.getSolde());

      compte1.crediter(200);
      compte2.debiter(300);
      System.out.println(compte1.getSolde());
      System.out.println(compte2.getSolde());
   }
}
```

Question 12. Indiquez comment compiler et exécuter cette classe en ligne de commande.

1.4 Évolution de la structure

Question 13. Ajoutez la méthode `List<Compte> getComptesFor(Client c)` à la classe `Agence`, qui permet de renvoyer l'ensemble des comptes associés à un client.

Question 14. Avec l'implémentation proposée jusqu'ici, la méthode `getComptesFor` est-elle efficace ?

Question 15. Quels changements proposeriez-vous à la classe `Agence` afin de mieux structurer ses données et résoudre les problèmes évoqués à la question précédente ?

Annexe : Utilisation d'`ArrayList`

La classe `ArrayList` du paquetage `java.util` permet de manipuler des tableaux de taille dynamique. `java.util.List` est une interface implémentée (entre autres) par `Vector`, `ArrayList`, `LinkedList`, ... Voici un exemple d'utilisation de la classe `ArrayList` et de l'interface `List` :

```
List<String> tab = new ArrayList<String>(); // contient des chaînes
System.out.println(tab.size()); // affiche 0 : la liste est vide

tab.add("hello"); // ajoute la chaîne "hello" (position 0)
System.out.println(tab.size()); // affiche 1 : un élément

String s = tab.get(0); // récupère le premier élément
System.out.println(tab); // affiche [hello]

tab.add("world"); // ajouté en fin de tableau (donc en position 1)
System.out.println(tab.get(0)); // affiche le premier élément

for (String elt : tab) { // parcourir tous les éléments
  System.out.println("Element : " + elt); // affiche l'élement courant
}

boolean b = tab.contains("world"); // b est vrai ici

tab.remove(0); // retire le premier élément et décale le suivant

System.out.println(tab.size()); // taille 1 ("world" en position 0)
```

```
System.out.println(tab); // affiche [world]

tab.clear(); // retire tous les éléments
System.out.println(tab.size()); // taille 0
```

Nous vous conseillons de consulter la documentation en ligne (documentation de l'API Java) qui donne plus d'informations sur la classe `ArrayList` : https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html.