TD – Modélisation d'une application

Utiliser la classe ArrayList

Les tableaux classiques sont disponibles en Java (comme en C++). L'API Java fournit d'autres classes permettant le stockage d'une collection d'objets, comme la classe ArrayList<>.

En fait, nous le verrons par la suite, mais l'interface java.util.List est implémentée sous la forme de ArrayList, LinkedList, Stack, …

On déclare de manière systématique, List<Toto> liste = new ArrayList<Toto>() afin de pouvoir changer l'implémentation de List si on le désire. Ici, nous utiliserons une classe particulière ArrayList

La classe ArrayList<Type> possède 2 avantages sur le tableau classique :

- elle ne requiert pas, a priori, la connaissance du nombre maximal d'éléments à mémoriser[1]
- elle permet la mémorisation d'objets de différentes classes dans un même tableau[2]

Elle propose, entre autres, les méthodes suivantes :

- void add(un objet) : ajoute un objet à la collection (à la fin)
- boolean remove(un objet) : enlève l'objet de la collection
- get(indice) : renvoie l'objet de la collection situé à une certaine position (indice)
- boolean contains(Object o) : retourne vrai si l'objet o est présent dans la collection
- int size() : retourne le nombre d'objets dans la collection

L'exemple suivant

```
import java.util.ArrayList;

public class Test {
```

1alors qu'un tableau classique requiert une taille maximale, sans possibilité d'extension : un objet de la classe ArrayList peut donc stocker un nombre "infini" d'objets (dans la limite de d'espace mémoire disponible)

2A lors qu'un tableau classique déclare le type des éléments mémorisés, il est possible dans un ArrayList de stocker des objets de type Object, il faut préciser la classe de l'objet récupéré par transtypage (changement de type forcé)

```
    public static void main(String[] args){

1    List liste = new ArrayList<>();

2   liste.add("première chaîne");
3   liste.add(new String("une deuxième chaîne"));

    int i;
4    System.out.println("Longueur de la chaîne : " + ((String)
(liste.get(i))).length());


    }

}
```

– ligne**1** : déclare un ArrayList (collection) de chaînes de caractères
– lignes 2-3 : y ajoute 2 objets chaînes de caractères ; remarquer la deuxième forme d'ajout : on instancie un objet qu'on ajoute directement à la collection ;
– ligne 4 : introduit une boucle à partir de 0, et tant que l'indice est inférieur à la taille du vecteur
– ligne 5 : liste.get(i) : renvoie l'élément du tableau liste à la position d'indice i (cet élément est de classe Object)
– (String) liste.get(i) : l'élément récupéré est un objet de la classe String
– ((String) liste.get(i)).length() : pour l'objet de la classe String, on invoque sa méthode 'length' qui renvoie sa longueur

Java propose une extension, les types génériques, qui permet de définir explicitement le type des éléments d'une collection. L'exemple ci-dessous déclare et instancie un tableau qui contiendra uniquement des objets des classes définies à la déclaration à l'instanciation :

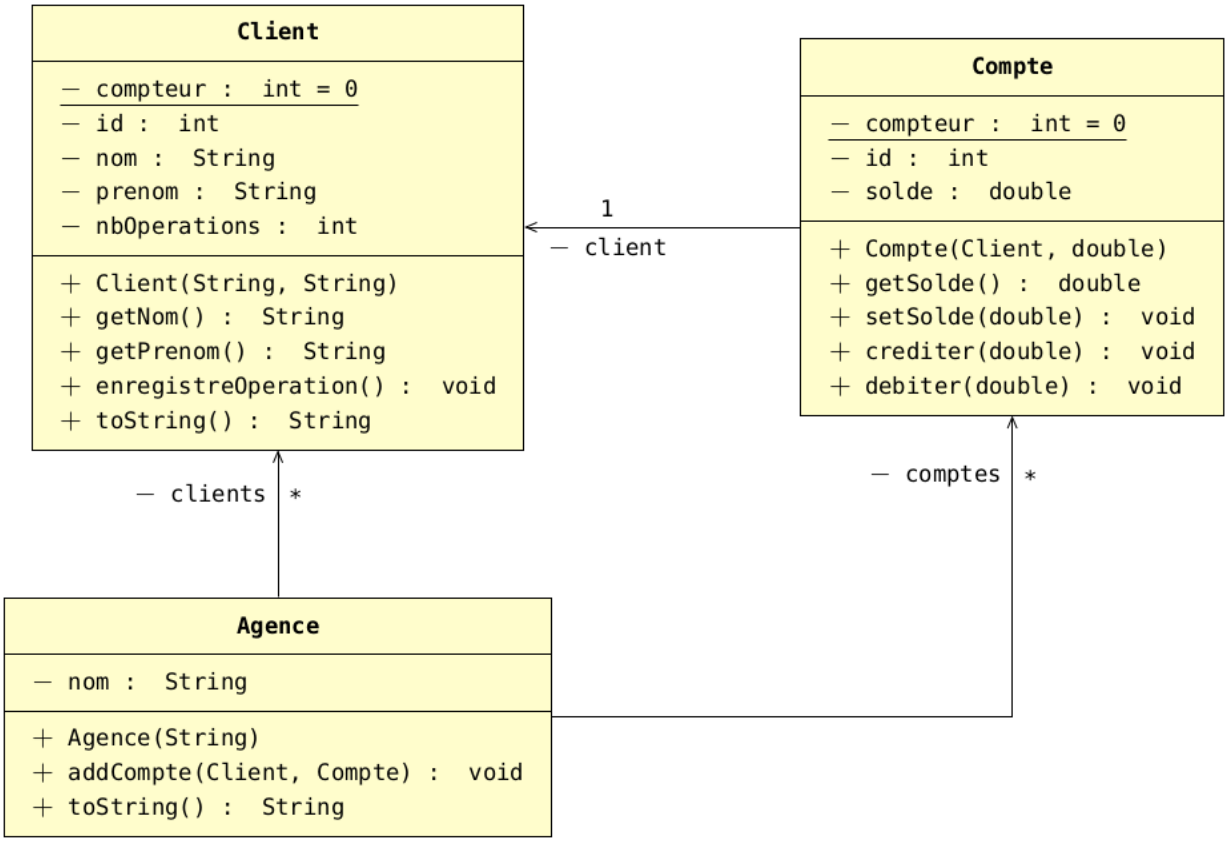
```
List<String> l1 = new ArrayList<String>(); //objets de classe String
 List<Polynome> l2 = new ArrayList<Polynome>(); //objets de classe Polynome
```

C'est ce deuxième type de déclaration que nous

Exercice - Application à la gestion d'une banque

Les classes suivantes, 'Compte', 'Client' vont permettre la création et la simulation du fonctionnement d'une banque : création des clients, ajout de comptes au clients et mouvements sur les comptes.

Inspiré d'un exercice sur la gestion d'une banque (L3 Sorbonne – Université)


Client
− compteur : int = 0
− id : int
− nom : String
− prenom : String
− nbOperations : int
+ Client(String, String)
+ getNom() : String
+ getPrenom() : String
+ enregistreOperation() : void
+ toString() : String
Compte
− compteur : int = 0
− id : int
− solde : double
+ Compte(Client, double)
+ getSolde() : double
+ setSolde(double) : void
+ crediter(double) : void
+ debiter(double) : void
1
− client
− clients *
− comptes *
Agence
− nom : String
+ Agence(String)
+ addCompte(Client, Compte) : void
+ toString() : String

Les délégations représentent des attributs de classes. Par exemple :

- comptes est un attribut private (identifié par le -) 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.

Pour se familiariser un peu avec les package, les trois classes que nous allons créer seront dans le package exercices.banques.

- Dans quel répertoire seront ces trois classes ?
- Quel est la commande (sur la ligne de commandes) pour compiler ces trois classes ?

Ci-dessous l'implémentation de la classe Client

Structure de la classe Client

- Quel est le nom de la classe ?
- Quel est son package ?
- Où doit se situer le fichier source de la classe ?
- Nommez les différentes éléments de la classe
  - attributs
  - constructeurs
  - méthodes
- Il y a deux méthodes permettant de tester l'égalité des clients. Essayez d'expliquer le principe de fonctionnement de la méthode equals, de la méthode clientEgal. Quel(s) est(sont) les avantages d'une méthode par rapport à l'autre ?
- Comment est calculé l'identifiant de chaque client ? Donnez les identifiants (id) des trois premiers clients créés
-

Classe Compte

La classe Compte garde en référence un client, elle contient le solde du compte.

- Proposez une implémentation Java conforme au diagramme UML
- Proposez une implémentation pour l'attribution de l'identifiant
- Les concepteurs demandent de comptabiliser les opérations sur le compte : vous devez donc incrémenter le compteur d'opérations du client à chaque crédit ou débit de son compte.
- Il est très classique de posséder plusieurs comptes dans une banque par exemple un compte courant et un compte d'épargne, mais un client peut même avoir jusqu'à 4/5 comptes. Comment faire pour prendre en compte cette multiplicité de comptes ?
- Proposez une implémentation de la méthode equals en s'inspirant de la classe Client

Classe Agence

Pour stocker les clients, nous avons à notre disposition principalement les List qui seront dérivées en ArrayList et les tableaux classiques. Nous avons vu les tableaux, les ArrayList sont présentées au début de ce TD.

- Donnez en quelques mots les différences entre tableaux et ArrayList. Donnez les attributs et le constructeur de la classe Agence dans les deux cas. Quelle solution préconisez-vous ?
- Proposez une implémentation de la classe Agence en utilisant les ArrayList. Attention, l'agence ne doit pas stocker de doublons, c'est-à-dire qu'un compte et/ou un client ne doit être enregistré qu'une unique fois

Voici le code situé dans le package test qui nous permet de tester notre code :

- Quel est le nom du fichier ?
- Où doit se situer le code de cette classe de test ?
- Dessinez l'arborescence des répertoires et des fichiers
- Quelle la ligne de commande pour compiler ce fichier ?
- Quel est la ligne de commande pour exécuter ce fichiers

```
package exercices.test;
import exercices.banque.Agence;
import exercices.banque.Client;
import exercices.banque.Compte;

public class TestBanque {

        public static void main(String [] args){
        Agence agence = new Agence("Société Géniale");
        Client client = new Client("Boniface", "Maurice");
        Client client2 = new Client("xxx", "Maurice");
        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());
```

```
        }


}
```

Exercice 3 - Application à la gestion d'une banque

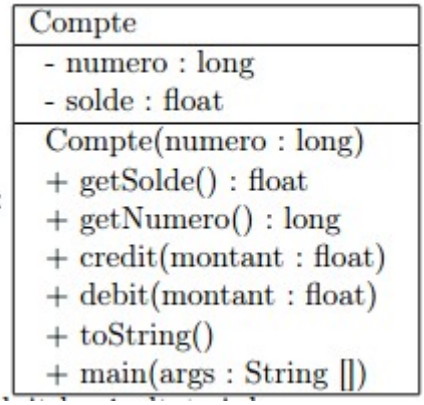
Les classes suivantes, 'Compte', 'Client' vont permettre la création et la simulation du fonctionnement d'une banque : création des clients, ajout de comptes au clients et mouvements sur les comptes.

Gérer les comptes

A réaliser

Créer la classe 'Compte' en utilisant le diagramme de classes et les précisions concernant les méthodes :

Classe Compte :



1. le constructeur mémorise le numéro de compte et remet le solde à 0
2. les méthodes accesseurs renvoient les valeurs des attributs correspondants
3. les méthodes 'credit' et 'debit', respectivement crédite et débite le solde du compte d'un certain montant
4. la méthode toString renvoie une représentation textuelle de l'objet (cf. exemple de résultat plus bas)
5. la méthode main définit les instructions suivantes :
   - instancier un compte de numéro 123456
   - afficher le compte (en utilisant la méthode toString() pour récupérer sa valeur)
   - créditer le compte de 1000
   - afficher le compte
   - débiter le compte de 1500
   - afficher le compte

Compiler et tester

L'exécution produit le résultat ci-dessous :

```
Compte 123456 - Solde : 0
Compte 123456 - Solde : 1000
Compte 123456 - Solde : -500 ** a surveiller **
```