BPOO - Sujet TD 7

Dut/Info-S2/M2103

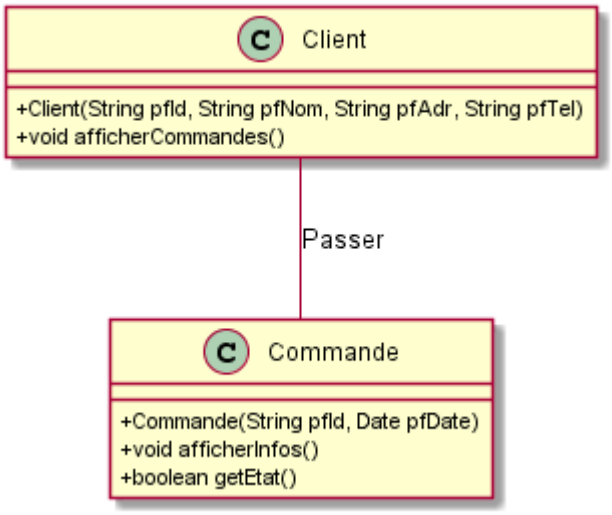
Table des matières

PreReq	Diagramme de classes, associations. Classes java. ArrayList
ObjTD	**Définir une association en java**.
Durée	**1 séance** de 1,5h

1. Domaine d'étude

On va s’intéresser à mettre en oeuvre de façon simple l’association suivante.

Diagramme UML des classes mises en oeuvre (sans multiplicités, définies ensuite)



Le but est de comprendre comment mettre en place des **associations navigables dans les deux sens**, **les traitements impliqués** et les limites existantes.

---

2. Principe

En TP, nous avons déjà travaillé le cas de l’association orientée (navigable dans un seul sens) ou agrégation dans le cas de la classe `AgenceBancaire` :

- la classe `AgenceBancaire` contient une ArrayList de `Compte`,
- les objets `Compte` **ne sont pas** reliés à une `AgenceBancaire`.

Pour mettre en oeuvre une association **navigable**, le principe est toujours de :

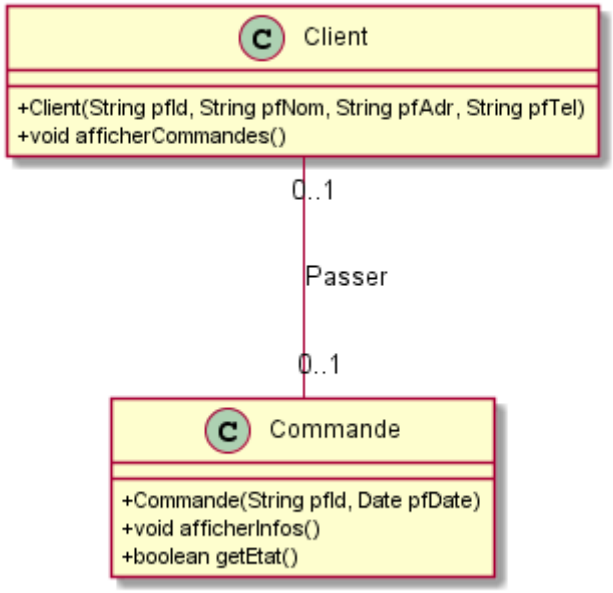
- une association UML se traduit dans un programme par des "liens" entre objets (java : référence(s) vers d’autre(s) objet(s)).
- pour la navigabilité : stocker les "liens" d’une classe vers une autre par un attribut, et ce dans les deux classes,
- selon la multiplicité : l’attribut aura un type particulier : objet, tableau/ArrayList, …,
- il faudra créer des méthodes permettant de maintenir les "liens" en "état cohérent" par rapport aux multiplicités.

---

3. Premier cas : association 0..1 — 0..1

On fixe les multiplicités de l'association "Passer" à "0..1" et "0..1" comme indiqué sur le diagramme ci-dessous.

Diagramme UML des classes mises en oeuvre (multiplicités 0..1)



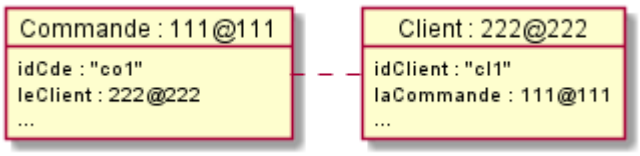
Pour réaliser cette solution, il faut ajouter **a minima** :

- à la classe Commande :
  - un attribut `leCient` de type Client : Client associé à la Commande,
  - l'attribut `leCient` vaut null si la Commande n'est associée à aucun Client,
  - deux méthodes :
    - `Client getClient ()` : permet d'obtenir le Client associé à la Commande (null si pas associée à un Client),
    - `void setClient (Client cli)` : permet de faire un lien entre la Commande et `cli` ; défait le lien lorsque `cli` vaut null.
- à la classe Client :
  - un attribut `laCommande` de type Commande : Commande associée au Client,
  - l'attribut `laCommande` vaut null si le Client n'est associé à aucune Commande,
  - deux méthodes :
    - `Commande getCommande ()` : permet d'obtenir la Commande associée à un Client (null si pas associé à une Commande),
    - `void setCommande (Commande comm)` : permet de faire un lien entre le Client et `comm` ; défait le lien lorsque `comm` vaut null.

Deux objets Client cl1 et Commande co1 sont associés si et seulement si :

- l'attribut `laCommande` de cl1 référence co1,
- l'attribut `leClient` de co1 référence cl1.

Exemple :



3.1. Une version incomplète : des liens cassés sont possibles

Cela donne la première version comme donnée sur la feuille jointe.

Soit le programme :

```
Commande com = new Commande ("com", "01/01/2014");
Client client1 = new Client ("client1", "c1", "ad1", "tel1");
Client client2 = new Client ("client2", "c2", "ad2", "tel2");

client2.setCommande (com);
com.setClient (client2);
client1.setCommande (com);
```

Cette solution simpliste sans aucun contrôle mène le programme ci-dessus à des liens incohérents entre objets Commande et objets Clients.

1. **Question 1** : Où est l'incohérence ? Dessinez les liens entre objets mis en place.
2. **Question 2** : Donnez des exemples d'autres erreurs potentielles.

3.2. Une version cohérente : maintien en cohérence des liens

Pour résoudre le problème et maintenir les liens cohérents : il faut que lors de l'association d'une Commande co à un Client cl :

- lier la Commande co au Client cl,
  - si le Client cl était déjà associé une Commande, l'association de cette Commande vers cl doit être rompue,
- lier le Client cl à la Commande co,
  - si la Commande co était déjà associé à un Client, l'association de ce Client vers co doit être rompue.

Il en est de même lorsqu'un Client est associé à une Commande.

Question : Pour réaliser ce traitement, écrire :

- dans la classe Commande :
  - une nouvelle méthode `void associerClient (Client cli)`, qui créera correctement l'association,
  - une nouvelle méthode `void desassocierClient ()`, qui défera correctement l'association,
- Dans la classe Client :
  - une nouvelle méthode `public void associerCommande (Commande comm)`, qui créera correctement l'association,
  - une nouvelle méthode `public void desassocierCommande ()`, qui défera correctement l'association.

Dans la classe Commande, la méthode `void associerClient (Client cli)` réalise le traitement :

- Si le client `cli` est déjà associé à une Commande (appelons la `com`) : défaire cette association (lier `com` à null (setClient())).
- Si la Commande est déjà associée à un Client : défaire cette association (lier ce Client à null (setCommande())).
- Créer les liens pour associer la Commande et `cli` (setClient() et setCommande()).

Dans la classe Commande, la méthode `void desassocierClient ()` réalise le traitement :

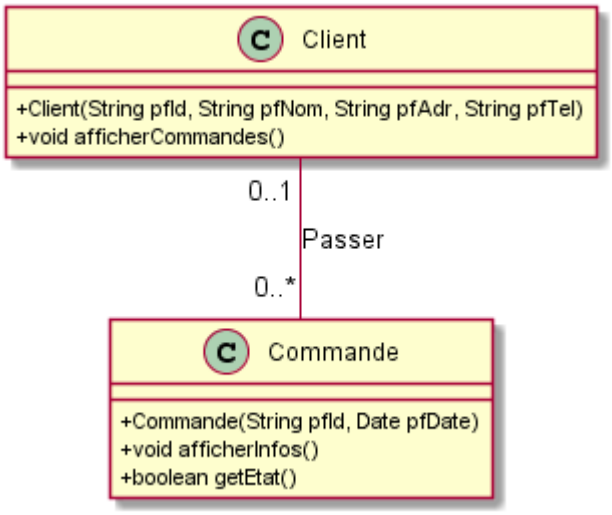
- Si la Commande est déjà associée à un Client : défaire cette association (lier ce Client à null (setClient())).
- Lier la Commande à null (setClient()).

---

4. Deuxième cas : association 0..1 — 0..*

On fixe les multiplicités de l'association "Passer" à "0..1" et "0..*" comme indiqué sur le diagramme ci-dessous.

Diagramme UML des classes mises en oeuvre (multiplicités 0..1)



Pour réaliser cette solution, il faut modifier les classes de départ **a minima** :

- à la classe Commande (pas de changement) :
  - un attribut `leCient` de type Client : Client associé à la Commande,
  - l'attribut `leCient` vaut null si la Commande n'est associée à aucun Client,
  - deux méthodes :
    - `Client getClient ()` : permet d'obtenir le Client associé à la Commande (null si pas associée à un Client),
    - `void setClient (Client cli)` : permet de faire un lien entre la Commande et `cli` ; défait le lien lorsque `cli` vaut null.
- à la classe Client :
  - un attribut `lesCommandes` de type ArrayList<Commande> : liste des objets Commande associés éventuellement au Client (0 éléments si pas de Commande),
  - trois méthodes :
    - `void addCommande(Commande comm)` : lie le Client à la nouvelle Commande `comm` : ajoute la Commande `comm` à `lesCommandes`,
    - `void removeCommande (Commande comm)` : enlève le lien du Client à la Commande `comm` : retire la Commande `comm` de `lesCommandes`,
    - `public boolean existeCommande(Commande comm)` : permet de savoir si le Client est associé à une Commande `comm` (`comm` est présent dans `lesCommandes`).
    - on aurait pu imaginer un "getter" de la liste des Commande auxquelles le Client est associé.

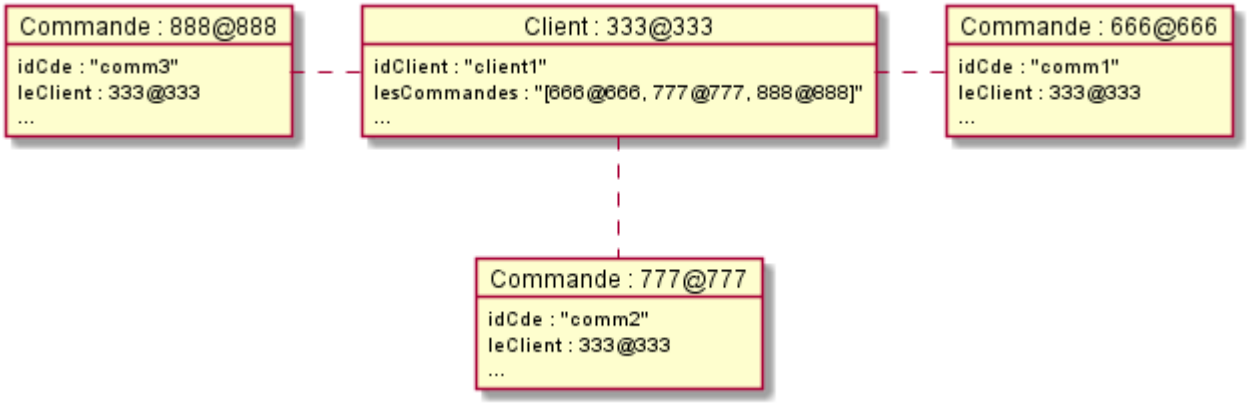
Cela donne la première version comme donnée sur la feuille jointe.

Deux objets Client cl1 et Commande co1 sont reliés si et seulement si :

- l'attribut `lesCommandes` de cl1 référence co1,

- l'attribut `leClient` de co1 référence cl1.

Exemple :



Question : Ecrire :

- dans la classe Commande, une nouvelle méthode `void associerClient (Client cli)`, qui créera correctement l'association,
- dans la classe Commande, une nouvelle méthode `void desassocierClient ()`, qui défera correctement l'association,
- dans la classe Client, une nouvelle méthode : `void associerCommande (Commande comm)`, pour créer une association,
- dans la classe Client, une nouvelle méthode : `void desassocierCommande (Commande comm)`, pour défaire une association.

Ces quatre méthodes doivent permettre de construire et défaire correctement les liens entre objets afin de maintenir les associations cohérentes (éviter les problèmes rencontrés dans le cas précédent).

---

5. Remarques finales

Toutes les méthodes sont déclarées publiques et donc sont accessibles aux clients, en particulier `setClient()`, `setCommande`, et `addCommande()/removeCommande()` en version 0..*.

Notons que les clients de la classe DOIVENT maintenant utiliser UNIQUEMENT les méthodes `associerClient()` et `associerCommande()` (et `desassocierCommande()` en version 0..*).

Pour cela 3 solutions :

- Documenter clairement chaque classe et chaque méthode pour en expliquer l'usage.
- Utiliser des visibilités autres que public qui résolvent le problème ; lié au langage et non étudié à ce jour.

- Trouver des algorithmes pour `associerClient()` et `associerCommande()` ne réclamant pas les méthodes setters qui posent problème ; complexe à mettre en oeuvre mais possible.

Enfin notons que les méthodes `associerClient()` et `associerCommande()` (et `desassocierCommande()` en version 0..*) devraient se nommer `passeePar (Client c)` dans la classe Commande et `passer(Commande c)` dans la classe Client.

Dernière mise à jour 2017-04-05 12:26:34 CEST