BPOO - Sujet TD 6

Dut/Info-S2/M2103

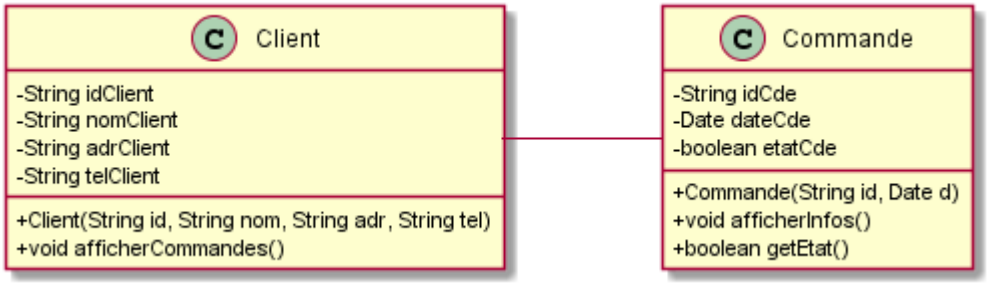
Table des matières

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

1. Domaine d'étude

On va s'intéresser à mettre en oeuvre de façon simple l'association suivante (simplifiée ici à l'extrême).

Diagramme UML des classes mises en oeuvre (sans multiplicités)



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

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, le principe sera toujours de :

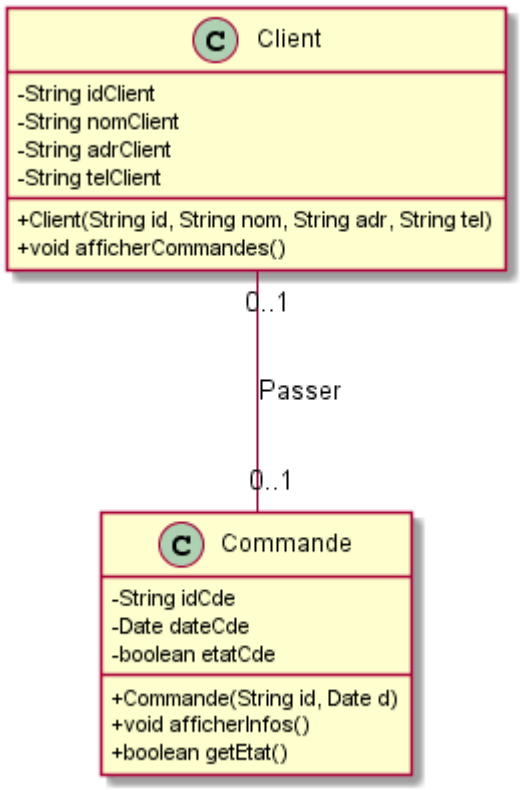
- stocker les "liens" d'une classe vers une autre par un attribut, et ce dans les deux classes,
- selon la multiplicité : l'attribut se voit changer de type (objet, tableau/ArrayList, …),
- il faudra créer des méthodes permettant de maintenir les relations en "état cohérent" par rapport aux multiplicités.

---

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

On fixe les multiplicités de l'association "Passer" à "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 éventuellement à la Commande (null si pas de Client),
  - deux méthodes :
    - `Client getClient ()` : permet d'obtenir le client associé à une Commande (renvoie null si

pas de Client),

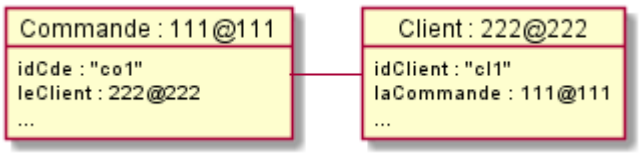
- `void setCommande (Commande comm)` : attribue une commande à un Client,

- à la classe Client :
  - un attribut `laCommande` de type Commande : Commande associée éventuellement au Client (null si pas de Commande),
  - deux méthodes :
    - `Commande getCommande ()` : permet d'obtenir la commande d'un Client (renvoie null si pas de Commande),
    - `void setCommande (Commande comm)` : attribue une commande à un Client,

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

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

Exemple :



2.1. Une version incomplète

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");

client1.setCommande (co1);
client2.setCommande (co1);
co.setClient (client2);
```

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.

2.2. Une version cohérente : maintient des liens

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

- si le Client est déjà associé une Commande, l'association de cette Commande vers ce Client soit rompue,
- si la Commande affectée est déjà associée à un Client, cette association soit rompue,
- associer la Commande affectée au Client,
- associer le Client à la Commande

Question : Ecrire :

- dans la classe Commande une nouvelle méthode `void definirClient (Client cli)`,
- dans la classe Client, une nouvelle méthode duale de fonctionnement similaire : `public void definirCommande (Commande comm)`.

La méthode `void definirClient (Client cli)` dans la classe Commande :

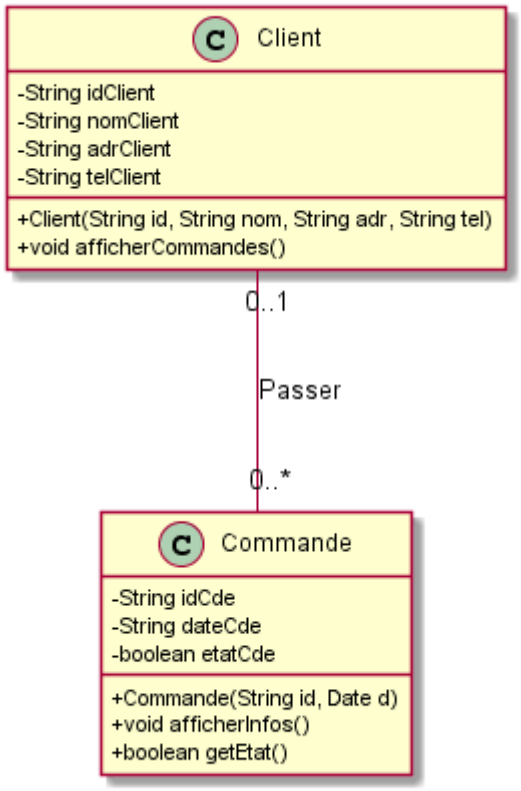
- Vérifie si le paramètre cli est null : auquel cas ne rien faire.
- Vérifie que la Commande ne soit pas déjà liée à ce Client cli : auquel cas ne rien faire.
- Si le Client cli est associé à une Commande : défaire cette association (associer cette Commande au Client null (setClient(null))).
- Si la Commande est déjà associée à un Client : défaire cette association (associer ce Client à la Commande null (setCommande(null))).
- Associer la Commande et le Client cli.

---

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

On fixe les multiplicités de l'association "Passer" à "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 modifier les classes de départ **a minima** :

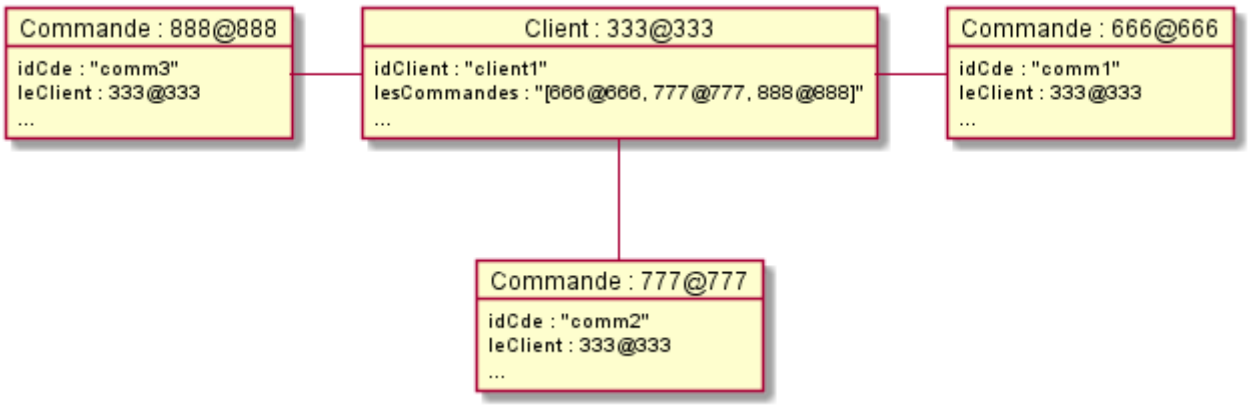
- à la classe Commande :
  - un attribut `leCient` de type Client : Client associé éventuellement à la Commande (null si pas de Client),
  - deux méthodes :
    - `Client getClient ()` : permet d'obtenir le client associé à une Commande (renvoie null si pas de Client),
    - `void setCommande (Commande comm)` : attribue une commande à un Client,
- 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)` : ajoute la Commande comm à `lesCommandes`
    - `void removeCommande (Commande comm)` : retire la Commande comm de `lesCommandes`
    - `public boolean existeCommande(Commande comm)` : permet de savoir si un objet Commande comm est présent dans `lesCommandes`

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 definirClient (Client cli)`,
- dans la classe Client, une nouvelle méthode duale : `public void ajouterUneCommande (Commande comm)`.

Ces deux méthodes doivent permettre de construire les liens entre objets afin de maintenir les associations cohérentes.

La méthode `void definirClient (Client cli)` dans la classe Commande :

- Vérifie si le paramètre cli est null : auquel cas ne rien faire.
- Vérifie que la Commande ne soit pas déjà liée à ce Client cli : auquel cas ne rien faire.
- Vérifie que le Client cli n'est pas déjà associé à cette Commande (existeCommande()).
- Si la Commande est déjà associée à un Client : défaire cette association (enlever la Commande au Client (removeCommande)).
- Associer la Commande et le Client cli.

La méthode `public void ajouterUneCommande (Commande comm)` dans la classe Client :

- Vérifie si le paramètre comm est null : auquel cas ne rien faire.
- Vérifie que la Commande comm ne soit pas déja enregistrée comme Commande de ce Client : auquel cas ne rien faire (existeCommende()).
- Si la Commande comm est associé à un Client : défaire cette association (removeCommande()).
- Associer la Commande et le Client cli.

Dernière mise à jour 2014-03-20 03:00:17 CET