TD-TP : Relation bidirectionnelle **1x1** en Java

MISE EN PLACE

Etant données la classe `Client` dont les objets sont caractérisés par un `nom` et un `téléphone`, et la classe `Table` dont les objets sont caractérisés par un `numéroTable`. Lorsqu'un `Client` est affecté à une `Table`, c'est de façon exclusive. Pour une `Table` donnée, on doit connaître son `Client` de même que pour un `Client` donné on doit connaître sa `Table`.

Travail à faire

1. Donnez le schéma de classe UML d'une telle situation.
2. Créer un projet Java sous Eclipse, intitulé `3.RelationSymetrique1x1` dans lequel coder les classes `Table` et `Client`.
3. Coder un `main()` dans une classe `TesterRelationSymetrique1x1` qui crée un `client1 <Martin, 0101010101>`, un `client2 <Durand, 0202020202>`, une `table1 <t01>` et une `table2 <T02>`, puis qui affiche la valeur de ces objets via `toString()`.

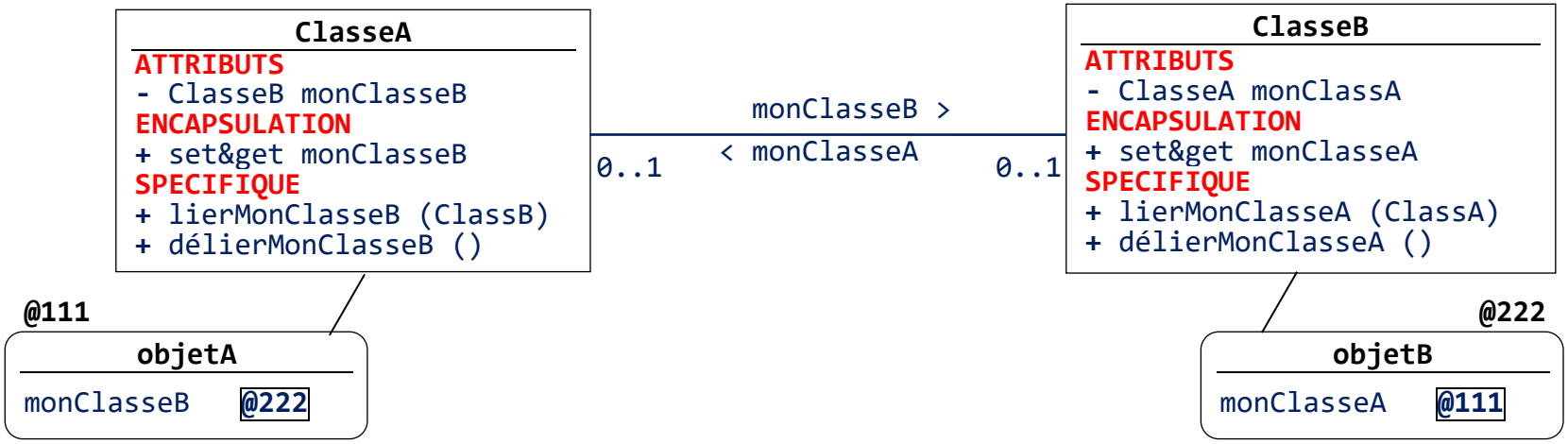
IMPLEMENTATION DE LA RELATION BIDIRECTIONNELLE 1x1

En fait, la relation modélisée avec UML induit l'obligation d'introduire des attributs et méthodes supplémentaires selon la navigabilité entre objets, voulue par le concepteur.

Le choix d'implémenter la navigabilité entre `ClassA` – `ClasseB` consiste pour le concepteur à se poser les deux questions suivantes :

- ***Un objet de `ClasseA` a-t-il besoin d'accéder à l'objet de `ClasseB` avec lequel il est lié ?***
  Si la réponse est **oui** alors il faut implémenter la navigabilité de `ClasseA` vers `ClasseB`.
- et de façon symétrique, ***Un objet de `ClasseB` a-t-il besoin d'accéder à l'objet de `ClasseA` avec lequel il est lié ?***
  Si a réponse est **oui** alors il faut implémenter la navigabilité de `ClasseB` vers `ClasseA`.

En cas de double **oui,** eh bien cette double navigabilité se modélise en UML, comme ci-dessous, qui en plus présente `objetA` et `ObjetB` qui sont liés.



Puisqu'en plus de mémoriser l'adresse de l'objet avec lequel il est lié (cf. `monClasseA` et `monClasseB`), il faut encapsuler et pouvoir `lier` deux objets avec une seule méthode qui prendra en charge la réciprocité des références croisées, tout en déliant les éventuels liens existants.

Travail à faire

4. Enrichir le schéma de classe de la réponse **1.** pour intégrer la double navigabilité entre `Table` et `Client`.

5. Modifier le code des classes `Table` et `Client` de sorte à intégrer les attributs et méthodes correspondantes, sachant que pour me `délier` « si je suis lié, le correspondant ne doit plus pointer vers moi et je ne dois plus pointer vers lui » et que pour me `lier` avec un nouveau correspondant « si mon futur correspondant existe, je dois me délier de mon correspondant actuel, délier mon futur correspondant de son éventuel lien, puis pointer vers lui et le faire pointer vers moi ».

6. Dans le `main`, tester votre solution :

- **a.** Créer un lien entre `table1` et `client1` de même qu'entre `client2` et `table2`
- **b.** Afficher via `toString()`, que vous aurez préalablement mis à jour, les objets `client1` et `table2` qui feront apparaître leur correspondant.
- **c.** Lier `client1` et `table2`
- **d.** Afficher `client1`, `client2`, `table1` et `table2` on devra observer que les relations du **a.** ont été supprimées au bénéfice de la relation **c.**