Mise en lumière

Dans ce document, j'ai décidé de mettre en avant le mécanisme d'instanciation des DAO que j'ai mis en place grâce au design pattern Factory, la réflexion et un fichier properties.

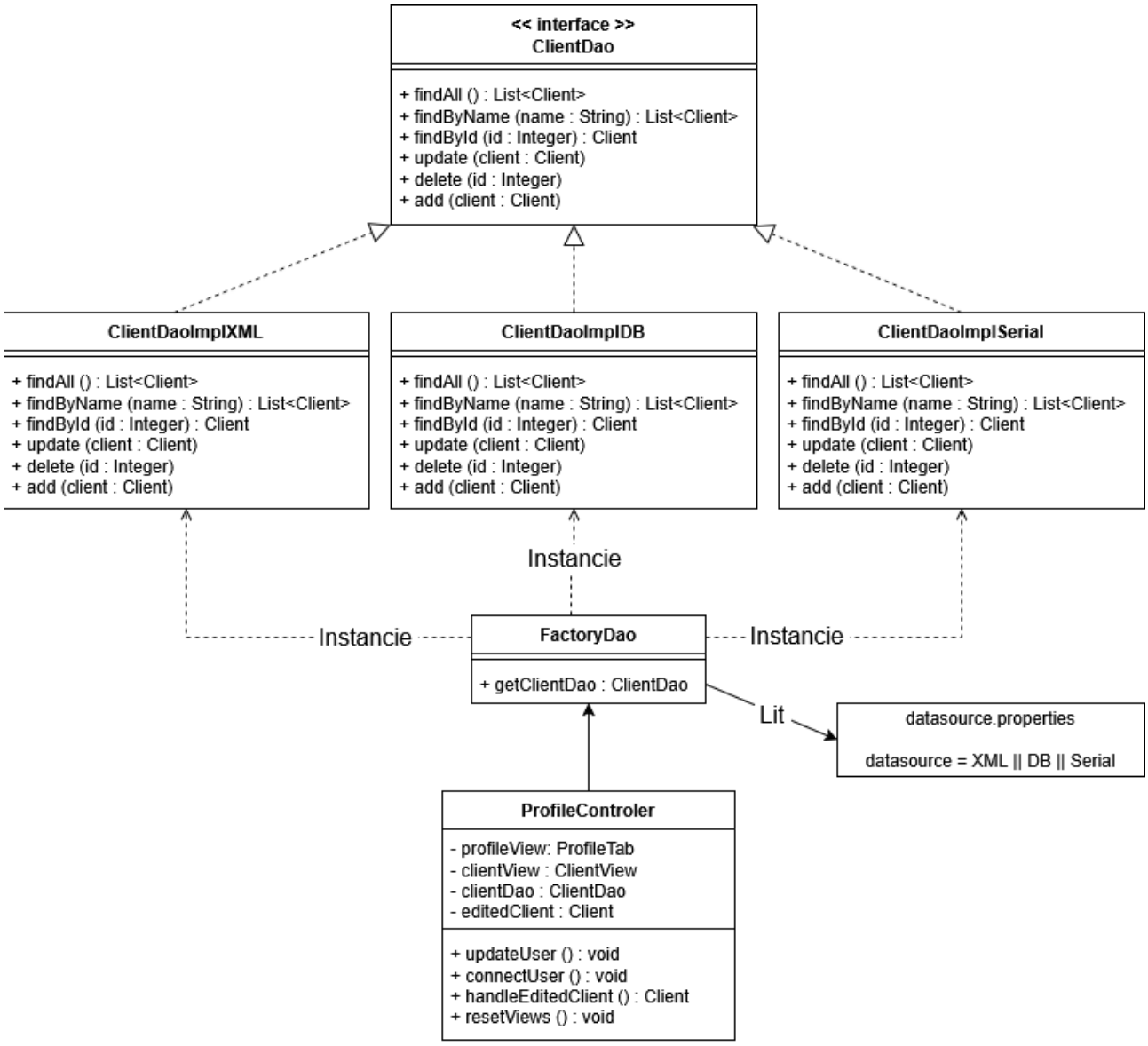
Le principe

Ce système permet de pouvoir modifier le type de source de données depuis lequel on veut récupérer les informations persistées.

Dans notre projet, la consigne étant d'utiliser une base de données SQL, l'intérêt d'implémenter ce mécanisme est plus dans un souci d'évolutivité de l'application. A l'avenir si l'on décide de récupérer les informations depuis une autre source tel un fichier sérialisé, un fichier XML, CSV, nous n'aurons pas de modifications majeure à apporter.

Une image valant mieux qu'un long discours, voici un diagramme UML qui reprend le mécanisme d'instanciation des DAO.

Les classes **ClientDaoImplXML** et **ClientDaoImplSerial** sont là à titre d'exemple, elles n'ont pas été implémentées dans le projet.



Comme on peut le voir, l'avantage de cette technique réside dans le fait que les contrôleurs n'ont plus la responsabilité de déterminer le type de source de données. On augmente ainsi d'un niveau supplémentaire le découplage entre l'application et le stockage des données.

Si demain on décide de changer la source de données, cela sera complètement transparent pour les contrôleurs. La seule chose à faire sera d'implémenter un nouveau DAO comme par exemple **ClientDaoImplXML** et de changer la configuration dans le fichier **datasource.properties**.

Le mécanisme

Tout se passe dans la classe **FactoryDao.java**.

Une première méthode *getDaoProps ( )* va lire le fichier **datasource.properties** et retourner sous forme d'une chaîne de caractères le type de datasource que l'on souhaite utiliser (« DB », « XML », « CSV », etc.).

Ensuite, on retrouve une méthode pour chaque DAO que l'on souhaite implémenter (Client, Beer, Order et OrderLine).

Chacune de ces méthodes repose sur le même principe :

1. On récupère le nom de la classe que l'on veut instancier avec la méthode Class.forName(*nom_de_la_classe)* et on l'enregistre dans l'objet *daoClass* de type Class. Le nom de la classe est construit avec une concaténation du début du nom de la classe + le type de datasource. Exemple :

```
"org.cnam.jbrasserie.dao.beer.BeerDaoImpl" + getDaoProps()
```

2. On récupère le constructeur de cette classe avec *daoClass.getConstructor( )* et on l'enregistre dans l'objet *constructor* de type Constructor.

3. À partir de là, on peut retourner une nouvelle instance de la classe avec la méthode *constructor.newInstance( )* . Cette méthode nous retournant un objet de type Object, il faudra le caster avec le type adapté comme par exemple *(ClientDao) constructor.newInstance( ).*

Du côté des contrôleurs, on se contente simplement d'appeler le DAO souhaité avec :

```
this.clientDao = FactoryDao.getClientDao();
```

Conclusion

J'avais découvert ce système en NFA035 et pour une raison que j'ignore j'avais trouvé fascinant le faite de pouvoir automatiser à ce point le changement de sources de données.

Ce système n'a rien d'innovant et ne m'a pas demandé tant de temps que cela à développer mais il me tenait à cœur de reproduire ce mécanisme dans ce projet.