Le Design Pattern DAO, la généricité, les Design Pattern Factory et Abstract Factory

La Généricité

Définition : définir des algorithmes identiques opérant sur des données de types différents.

Exemple en java List<T> où T représente type.

Les résolutions de types se font à l'interprétation.

Une fois instancié, l'objet de la classe ne pourra travailler qu'avec le type de données spécifié.

A l'instanciation, on pourra avoir : List<ClasseObjet> ou List<int>

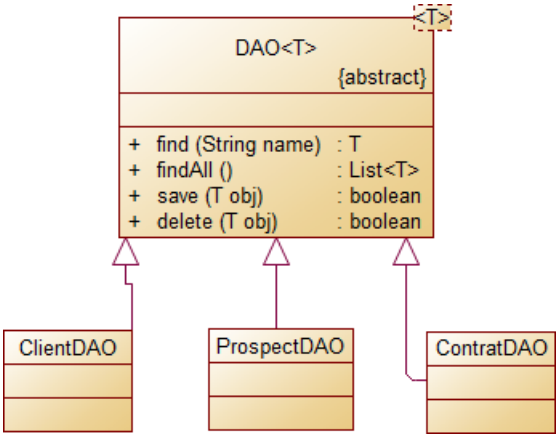
Rq : Integer est la classe wrapper (enveloppe) de int (idem pour Float float…). On appelle l'autoboxing la fonctionnalité permettant de transformer un objet de type primitif en un objet de sa classe wrapper.

Le Design Pattern DAO

C'est un pattern d'architecture qui utilise la généricité. Il est utilisé par le design pattern Factory et éventuellement le pattern Abstract Factory

Ce pattern permet de faire le lien entre la couche d'accès aux données et la couche métier d'une application.

Il permet de mieux maîtriser les changements susceptibles d'être opérés sur le système de stockage des données. Donc, par extension, de préparer une migration d'un système à un autre (BDD vers fichiers XML, par exemple…).



On utilisera la généricité en créant une classe générique.

La classe générique est une classe abstraite ou une interface qui accepte tous les types d'objets. Elle se caractérise par le signe <T>

Elle va permettre de faire du « polymorphisme de type » appelé aussi « paramétrage de type »

Dans le cadre d'une DAO, elle va avoir les méthodes abstraites qui correspondent à celles de vos classes DAOClient et DAOProspect : findAll(), find(), save() et delete(). On pourra l'appeler DAO.java

Si vous avez fait hériter vos classes DAOClient et DAOProspect d'une classe DAOSociete, vous devez créer une interface.

Sinon, il est toujours plus intéressant de créer une interface, cela laisse la porte ouverte à un éventuel héritage.

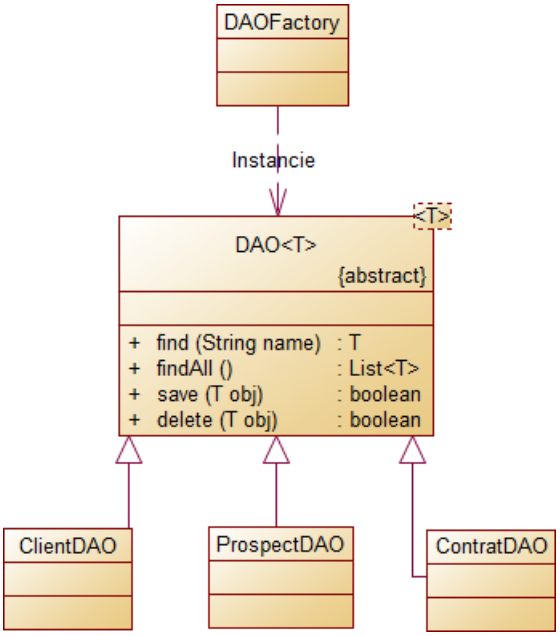
En faisant hériter ou en faisant implémenter vos classes DAOClient et DAOProspect de la classe ou de l'interface DAO, vous allez forcer ces classes à implémenter les méthodes de la classe (ou interface) DAO et ainsi forcer le polymorphisme.

Exemple : public class DAOCustomer extends DAO<Customer> { }

La généricité permet cela car cette classe (ou interface) DAO acceptera tous les types d'objets. On pourra donc rajouter d'autres fonctionnalités à notre application sans modifier cette classe.

Le Design Pattern Factory (Fabrication d'objets)

https://www.oracle.com/java/technologies/dataaccessobject.html



Il consiste à déléguer l'instanciation d'objets à une classe. Cela permettra, en le couplant avec le pattern Abstract Factory, de pouvoir switcher d'une persistance de données à une autre avec un simple changement de paramètre (cf la fin du cours)

➔ La classe DAOFactory
- Instancie les objets des classes DAO, à l'appel des méthodes getCLient, getProspect …

➔ La classe DAO
- Est une classe abstraite générique
- Possède des méthodes abstraites d'accès à la persistance de données

➔ Les classes DAOClient, DAOProspect, etc..
- Sont instanciées par la classe DAOFactory
- Héritent de la classe abstraite générique DAO
- Devront implémenter les méthodes de la classe abstraite DAO

Avantages :

- Maintenance simplifiée : l'application a les mêmes noms de méthodes pour toutes les classes DAO
- Evolution de l'applicatif : il est facile d'ajouter ou de supprimer des classes à la DAO sans avoir à modifier le reste de l'application
- Les objets DAO sont instanciés au même endroit, dans la classe DAOFactory

Le Design Pattern Abstract Factory

Ce Design Pattern va permettre de gérer plusieurs sources de persistance de données dans un applicatif sans avoir à modifier le reste de l'applicatif.

L'utilisateur pourra passer d'un mode de persistance à un autre, comme d'un gestionnaire de base de données à un autre sans modifier le reste de l'application