TIMMO PROJECT :

DOSSIER FINAL

Table des matières

[Principes de conception et choix d’implémentation 3](#_Toc501639257)

[Design patterns : 3](#_Toc501639258)

[Sous-typage : 3](#_Toc501639259)

[Types primitifs : 3](#_Toc501639260)

[Divisions en package : 3](#_Toc501639261)

[Gestion des Biens et Clients : 3](#_Toc501639262)

[Couplage faible : 3](#_Toc501639263)

[Format de l’application 4](#_Toc501639264)

[Use cases implémentés : 5](#_Toc501639265)

# Principes de conception et choix d’implémentation

URL de l'application : https://github.com/alex-jacquot/TimmoProject

## 

## Design patterns :

Singleton : Agence, BienManager, ClientManager

Factory : BienFactory, ClientFactory, Délègue la creation des objets de Client et Bien à une classe pour automatiquement créer les dépendances avec les autres classes du packages correctement ainsi que fournir un attribut id pertinent à l’objet.

Command: Fournit les outils pour faciliter l’interface entre le code métier/modèle et l’utilisateur et ainsi disposer d’un programme principal concis

## Sous-typage :

Pour avoir une structure facilement modulable, les sous-types de bien (Appartement, Maison et Terrain) sont traités de la même manière pour la classe Bien Manager mais sont créés différemment au sein de BienFactory.

Les sous-types concrets de Bien sont distingués par une variable d’objet énumération BienType.

## Le principe de substitution de Liskov est respecté dans certains cas.

## Types primitifs :

Pour limiter l’usage des types primitifs, plusieurs classes dont des énumérations ont été crées pour remplacer les fameux objets String

Pour éviter la duplication de code :

## Divisions en package :

L’application est divisée en 5 packages dont notamment le package destiné aux actions d’agence, un gestionnaire de biens et un de clients à accès uniforme pour privilégier le couplage faible, ainsi que les commandes et exceptions

## Gestion des Biens et Clients :

Les 2 classes disposent d’un id associé pour éviter les doublons (cause moins d’erreurs que d’utiliser directement un HashSet dans les classes stockant ces objets.

## Couplage faible :

Les Biens et Clients font tous 2 partie d’un package séparés. Pour effectuer des opérations sur un Bien ou un Client, l’Agence passe directement et uniquement par la classe Manager associée.

De même, au lieu d’avoir pour chaque Bien et chaque Client un attribut Mandat, l’Agence stocke directement les Mandats qui contiennent les références du Bien en vente et du Client propriétaire.

Les Classes métiers Client et Bien n’ont pas d’opérations autres que des getters/constructeurs.

Un Bien peut être stocké dans l’agence sans avoir un Mandat correspondant et donc sans être en vente.

De même un Client peut ne pas avoir de Mandat de vente et être simplement inscrit à l’Agence

## Javadoc

La javadoc des classes et méthodes est fournie dans le répertoire "doc"

## Format de l’application

Le fichier principal de l’application de test est le seul à ne pas être dans un package et ainsi le rend facile à identifier.

Le programme principal de test fourni des données de test utiles.

# Use cases implémentés :

## Implémentés

Créer un Bien

Créer un Client

Créer un mandat de vente

Liste les biens

Lister les Biens en vente

Lister les clients

## Non implémentés

La gestion des Vœux n’est pas implémentée car on peut de toute façon permettre à un utilisateur d’acheter un Bien qu’il a vu dans une liste.

Pour la même raison, la gestion des Publicités n’est pas prioritaire et on peut directement afficher les informations du Bien.

Les Rendez-Vous ne valaient pas spécialement la peine d'être implémentée