Projet Final POO - Rapport

Florian BAUDRON, Lucas ESTRADE, Luc FRANCOIS, Kévin PETIT

Modélisation UML :

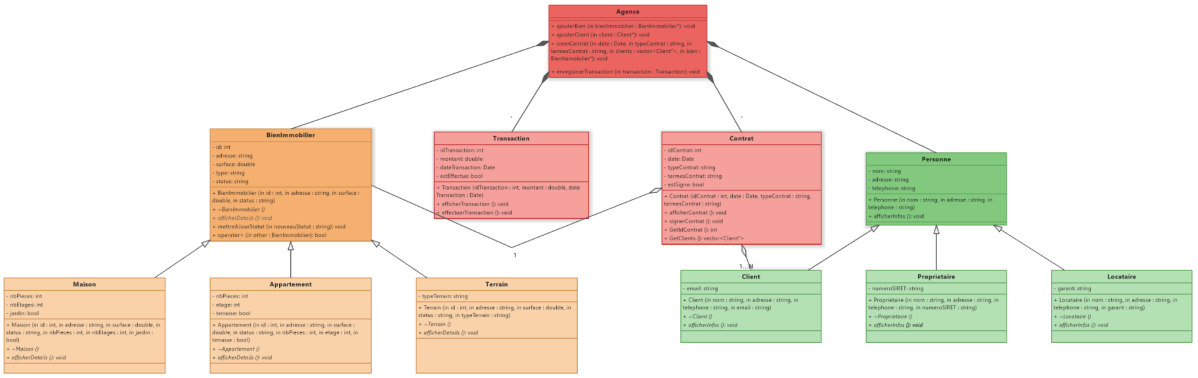

Image PNG non compressée disponible dans le rendu Moodle

Exercice 1 :

Dans cet exercice, nous définissons la classe ***BienImmobilier***, représentant un bien immobilier. Chaque bien est caractérisé par les éléments suivants :
- Un identifiant unique (nombre entier)
- Une adresse (chaîne de caractères)
- Une superficie (nombre décimal)
- Un type (chaîne de caractères)
- Un statut, qui peut être modifié au besoin et est représenté par une chaîne de caractères.

La classe propose trois méthodes principales :

- ***afficherDetails()***: Cette fonction permet d'afficher toutes les informations du bien immobilier, y compris son identifiant, son adresse, sa superficie, son type et son statut.
- ***mettreAJourStatut(string nouveauStatut)***: Une fonction qui prend en paramètre un nouveau statut sous forme de chaîne de caractères et le met à jour pour le bien immobilier concerné.
- un constructeur par paramètres pour initialiser les attributs

Nous avons également créé la classe ***Personne*** pour représenter les individus. Chaque personne est caractérisée par les éléments suivants :

- Un nom (chaîne de caractères)
- Un numéro de téléphone (chaîne de caractères)
- Une adresse (chaîne de caractères)

La classe ***Personne*** propose deux fonctions :

- Un constructeur prenant en paramètre les informations nécessaires pour initialiser les attributs de la personne.
- La méthode ***afficherInfos()***, qui affiche tous les attributs de la personne dans la sortie standard.
- Cette classe vise à fournir une représentation claire des informations essentielles liées à une personne, tout en permettant une manière simple d'afficher ces informations.

En outre, nous avons introduit la classe ***Contrat***, conçue pour représenter les contrats. Chaque contrat est décrit par les éléments suivants :

- Un identifiant de contrat, représenté par un entier (int).
- Une date, spécifiée à l'aide d'une classe *Date* spécialement créée.
- Un type de contrat, identifié par une chaîne de caractères.
- Les termes du contrat, détaillés également sous forme de chaîne de caractères.
- Un indicateur booléen, ***estSigne***, qui indique si le contrat a été signé ou non.

Les fonctions fournies dans cette classe sont les suivantes :

- Un constructeur permettant d'initialiser les attributs du contrat en utilisant des paramètres.
- La méthode ***afficherContrat()***, qui affiche toutes les valeurs des attributs du contrat.
- La méthode ***signerContrat()***, qui modifie le booléen ***estSigne*** pour indiquer que le contrat a été signé.

Exercice 2 :

Pour les classes héritées, nous avons étendu les attributs existants pour fournir des détails spécifiques à chaque type d'objet.

Pour la classe ***Maison*** :
En plus des attributs hérités de la classe ***BienImmobilier***, nous avons ajouté les éléments suivants :

- ***nbPieces*** : Nombre de pièces dans la maison.
- ***nbEtages*** : Nombre d'étages de la maison.
- jardin : Un booléen pour indiquer la présence d'un jardin.

Pour la classe ***Appartement*** :
En plus des attributs hérités de la classe ***BienImmobilier***, nous avons inclus :

- ***nbPieces*** : Le nombre de pièces dans l'appartement.
- ***etage*** : L'étage où se trouve l'appartement.
- terrasse : Un booléen pour déterminer si l'appartement dispose d'une terrasse.

Pour la classe ***Terrain*** :
Outre les attributs hérités de la classe ***BienImmobilier***, nous avons introduit :

- ***typeTerrain*** : Une spécification du type de terrain.

Pour la classe ***Client*** :
En plus des attributs hérités de la classe ***Personne***, nous avons ajouté :

- ***email*** : L'adresse e-mail du client.

Pour la classe ***Proprietaire*** :
En plus des attributs hérités de la classe ***Personne***, nous avons inclus :

- ***numeroSIRET*** : Le numéro SIRET du propriétaire.

Pour la classe ***Locataire*** :
En plus des attributs hérités de la classe ***Personne***, nous avons également intégré :

- ***garant*** : Le nom du garant du locataire.

Nous avons aussi redéfini toutes les méthodes d'affichage pour inclure les nouveaux attributs.

Exercice 3 :

Pour cet exercice, nous avons décidé de rendre les méthodes d'affichage virtuelles pour toujours afficher les bons attributs même si le pointeur pointe sur la classe de base.

Exercice 4 :

Pour cet exercice, nous avons décidé que dans le constructeur de la classe ***Contrat***, nous lançons des exceptions ***invalid_argument*** dans deux cas de figure : lorsque le paramètre ***termesContrat*** est vide et lorsque la date est invalide. Des commentaires adaptés sont renvoyés en conséquence.

Exercice 5 :

Dans cet exercice, nous avons surchargé l'opérateur '**>**' (supérieur strict) pour les classes ***BienImmobilier*** et ***Personne***, permettant ainsi de comparer des objets de ces classes en fonction de critères spécifiques.

Par la suite, nous avons créé un modèle de fonction nommé '***superComparison***' dans le fichier principal (***main***). Cette fonction prend deux paramètres et renvoie le maximum entre les deux en utilisant l'opérateur '**>**'.

Exercice 6 :

1) Dans la conception de la relation entre la classe ***Contrat*** et la classe ***Client***, nous avons introduit un vecteur de pointeurs vers des objets de type ***Client**** dans la classe ***Contrat***. Cela permet de représenter les clients associés à un contrat donné.

De plus, pour établir la relation entre la classe ***Contrat*** et la classe ***BienImmobilier***, nous avons ajouté un pointeur de type ***BienImmobilier**** dans la classe ***Contrat***. Ce pointeur représente le bien immobilier associé à ce contrat spécifique. Nous avons pris la décision qu'un seul bien immobilier peut être lié à un contrat à la fois.

Il est important de noter que ces relations sont des agrégations. Même si un contrat est rompu, les clients et le bien immobilier associés existent toujours.

2) Dans la classe ***Agence***, nous avons introduit deux vecteurs pour stocker les biens immobiliers et les personnes associées à l'agence. Ces vecteurs sont des associations de compositions, ce qui signifie que si l'agence est détruite, les biens et les personnes qui lui sont associés seront également détruits.

La composition implique une relation forte où les objets associés sont considérés comme faisant partie intégrante de l'objet parent. Ainsi, si l'agence est supprimée, tous les biens immobiliers et les personnes qui lui sont liés seront également supprimés, car ils dépendent de l'existence de l'agence pour leur propre existence.

Tests :

Dans le fichier principal (***main***), nous avons effectué tous les tests nécessaires pour vérifier le bon fonctionnement de chaque fonction que nous avons créée. Ces tests comprennent l'appel de toutes les fonctions avec différentes valeurs d'entrée pour s'assurer qu'elles produisent les résultats attendus.

Les classes ***BienImmobilier*** et ***Personne***, ainsi que leurs classes héritées, sont ainsi manipulées de manière extensive, afin de s'assurer du bon déroulement du programme quel que soit le type de bien immobilier (maison, appartement ou terrain) ou de personne (client, propriétaire ou locataire) manipulé, notamment lors de la création de contrat.

Quelques tests se concentrent également sur la gestion d'erreurs, vérifiant que le programme détecte correctement les cas particuliers et les traite en conséquence, sans empêcher le bon fonctionnement de la suite du programme principal.

Réflexions et améliorations futures :

Ce travail nous a montré l'application concrète et l'utilité de la plupart des concepts et fonctionnalités de la programmation orientée objet : encapsulation, héritage, polymorphisme, patrons de classes et de fonctions, surcharge d'opérateurs, utilisation de références et de pointeurs, gestion de mémoire (avec ***new*** et ***delete***), gestion des erreurs (avec ***try/catch/throw***)... Il nous a ainsi permis de revoir l'ensemble des acquis de cette session.

Si nous devions améliorer notre système, nous pourrions nous concentrer sur un développement plus approfondi de la gestion des transactions, qui reste assez simple dans cette version.

Nous pourrions imaginer également renforcer la détection d'erreur lorsque l'utilisateur complète des champs liés à des attributs spécifiques. Pour l'email du client ou le SIRET du propriétaire, on pourrait inclure une analyse de la chaîne de caractères entrée afin de vérifier qu'elle est bien conforme au format attendu.