Notion de Classe Canonique en C++

Constituants d'une Classe Canonique

On appelle une classe canonique toute classe qui contient à minima les 4 éléments suivants

- Au moins un constructeur régulier.
- Un constructeur de recopie.
- Un destructeur.
- Un opérateur d'affectation.

Exemple de modélisation UML :

MaClasse
ATTRIBUTS
CONSTRUCTEURS
MaClasse (*paramètres_éventuels*) // Constructeur régulier
MaClasse (const MaClasse& objetModeleAcopier) // Constructeur de recopie
DESTRUCTEUR
~MaClasse () // Destructeur
METHODES USUELLES : operator=
MaClasse& operator= (const MaClasse& objetDeMaClasse) // Affectation

La classe MaClasse *est une classe canonique.*

Même si le langage C++ propose un constructeur régulier par défaut, un constructeur de recopie par défaut, un destructeur par défaut et une opération d'affectation par défaut, il est souvent préférable de développer soi-même ces 4 méthodes.

En fait, dans les implémentations par défaut de ces 4 opérations, C++ se contente de gérer les valeurs contenues dans les attributs de l'objet à recopier. Or cela pose question, entre autres, lorsque les valeurs stockées dans les attributs de l'objet à recopier sont des adresses d'objets.

- *Faut-il recopier la même adresse dans le nouvel objet, qui de fait pointera au même endroit que l'objet modèle à imiter ?*
- *Ou bien faut-il créer des objets personnels vers lesquels pointer, tout comme l'objet modèle a ses propres objets sur lesquels il pointe ?*

La réponse réside dans la sémantique portée par le lien qui unit « objet qui pointe » et « objet pointé ». Selon la sémantique voulue et l'implémentation choisie, le constructeur de recopie de même que l'opérateur d'affectation créeront des objets personnels sur lesquels pointer, s'ils sont créés hors du nouvel objet (cf. ils ne sont pas 'embarqués') ; et le destructeur supprimera les objets personnels s'ils ne sont pas embarqués.

Exemple illustratif de Classe Canonique « Le projet Hotels »

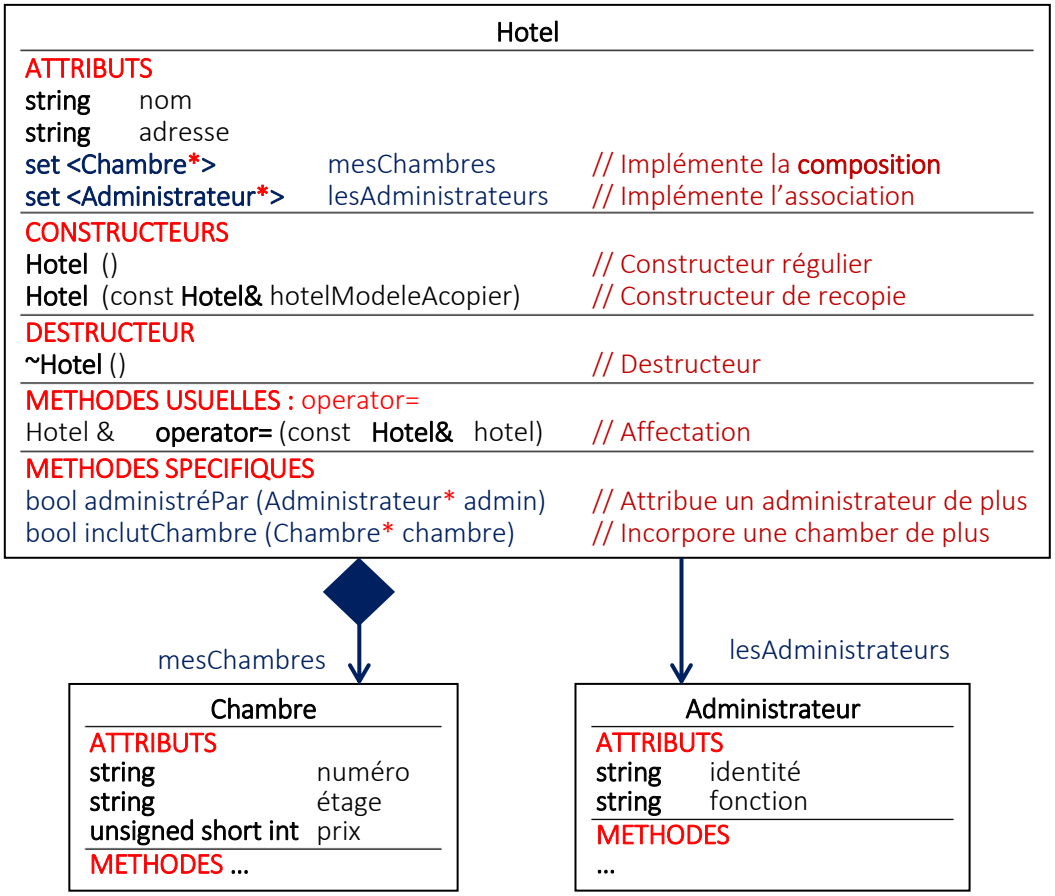
Cet exemple va mettre en évidence que les instances de la classe Hôtel, ont un attribut qui pointe sur des objets personnels non embarqués (Chambre), et ont un attribut qui pointe sur des objets non personnels (Administrateurs).

1. Règles de gestion

- Un Hôtel est défini par un nom et une adresse.
- Il est composé de Chambres définies par un numéro, un étage et un prix.
- Il est géré par des Administrateur définis par une identité et une fonction.

2. Analyse UML : une rapide analyse conduit au schéma UML ci-dessous ou apparaissent

- Une composition forte ◆, à savoir, si un hôtel disparait, ses chambres disparaissent,
- et une association simple ⟶, à savoir, si un hôtel disparait, ses administrateurs perdurent.



Modélisation UML

3. Décisions pour mémoriser les relations modélisées en UML

La relation lesAdministrateurs est implémentée par un ensemble d'éléments hors de l'objet (cf. set<Administateur*>). L'attribut lesAdministrateurs mémorise l'adresse de chacun des administrateurs.

La relation mesChambres pourrait être implémentée par un ensemble d'éléments embarqués dans l'objet (cf. set<Chambre>), mais par nécessité pédagogique, nous considérons que les chambres de l'hôtel seront hors de l'objet (cf. set<Chambre*>).

Ainsi, l'attribut mesChambres mémorise l'adresse de chacune de **ses** chambres.

4. Conséquences sur la gestion des relations

Constructeur de recopie : un hôtel créé sur la base d'un d'hôtel modèle existant, nécessitera la création de nouvelles chambres qui lui seront propres ; chambres sur lesquelles pointera ce nouvel hôtel.

Opérateur d'affectation : un hôtel qui se voit affecté les attributs d'un d'hôtel source existant, implique la création de nouvelles chambres qui lui seront propres ; chambres sur lesquelles il pointera.

Lors de la destruction d'un hôtel, il faut supprimer ses chambres mais pas ses administrateurs.

5. Tester votre solution avec un programme qui

1. Déclare un Hotel hotel ("LesDormeurs", "Terrain Instable, 75.000 PARIS")
2. Déclare deux administrateurs... admin1 ("Etcheverry", "President") et admin2 ("Dagorret", "Presidente") … qui administrent l'hotel
3. Déclare trois chambres chambre1 ("Campagne", "Rez-de-chaussee", 50), chambre2 ("Royale", "Etage 1", 100) et chambre3 ("Solaire", "Etage 2", 150) ... qui composent l'hotel
4. Crée un Hotel nommé hotelBis sur la base de l'hotel déjà existant (cf. plans et organisation). Pour cela invoquer le constructeur par recopie
5. Crée un Hotel hotelTer qui est une copie de hotel existant.
6. Pour cela invoquer l'opérateur d'affectation operator=
7. Modifie le prix de chambre dans hotelBis (cf. hotelBis.modifierPrixChambre("Campagne", 70)) et dans hotelTer (cf. hotelTer.modifierPrixChambre("Solaire", 170))
8. Modifie un attribut d'un administrateur
9. Rase / supprime le premier hotel - ce qui doit supprimer ses chambres, mais ne pas affecter les administrateurs
10. Vérifie l'intégrité des caractéristiques de hotelBis et de hotelTER

Solution Code::Blocks sera mise sur eLearn