Module M3105

Conception & Programmation Objet Avancées

Chargé de cours :

Guillaume Cleuziou

Organisation du module

- 1 feuille (TD&TP) par semaine
- Séances de TP progressives
- Test de restitution après chaque TP
- DS en fin de module
- Cours Célène

Contenu 1ère partie

- Conception orientée objets (UML en pratique)

Prérequis 1ère partie

- UML (diagrammes de base)
- Langage Objet (Java)
- Bases de données (Merise)

Modélisation statique

Diagramme de classes

Source : P. Roques « UML2 par la pratique », 5ème édition. Eyrolles.

Diagramme UML de classes

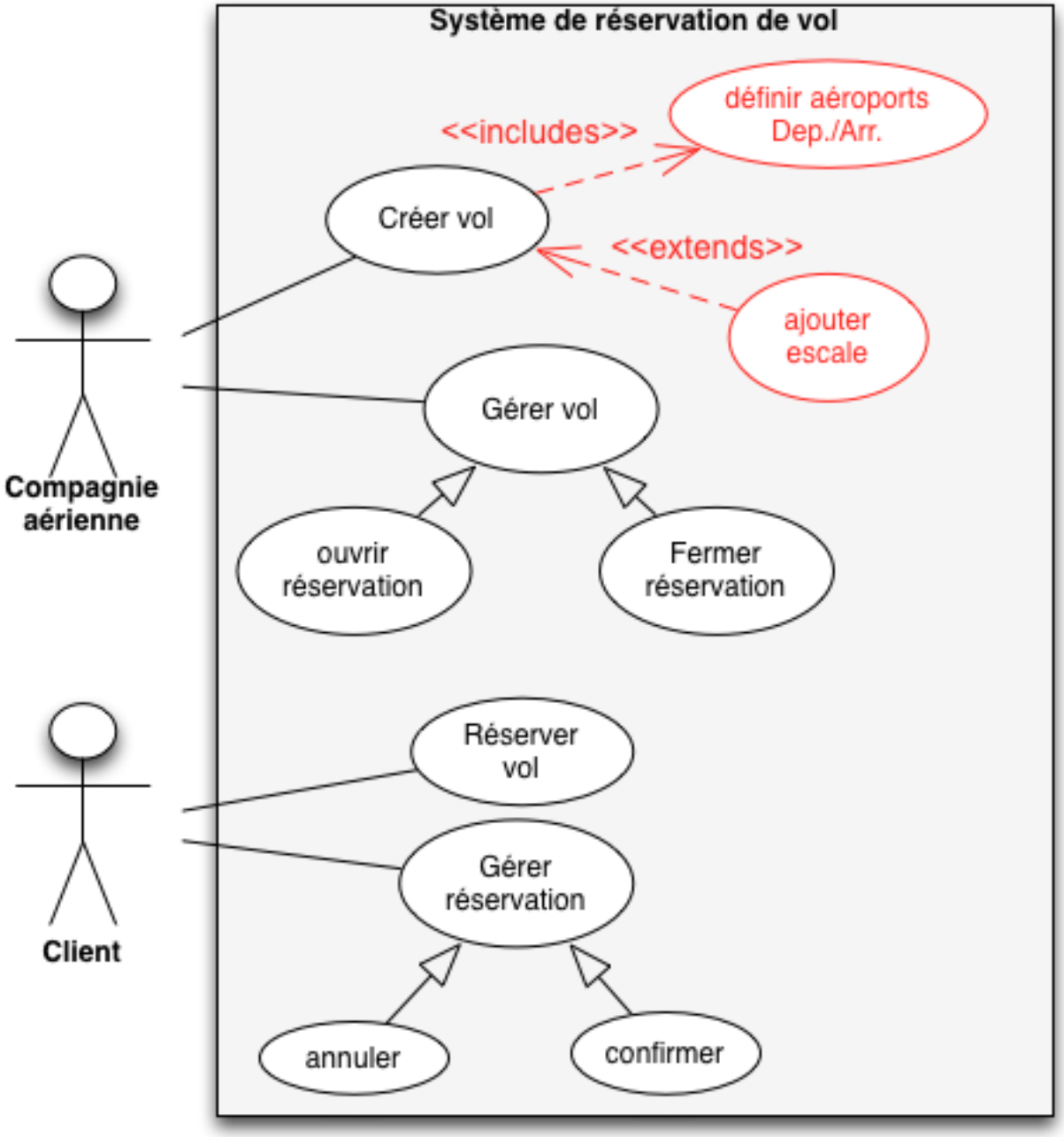
- Ça sert à quoi ? Ça contient quoi ?

Décrire la structure statique d'un programme (objet)

...en termes de

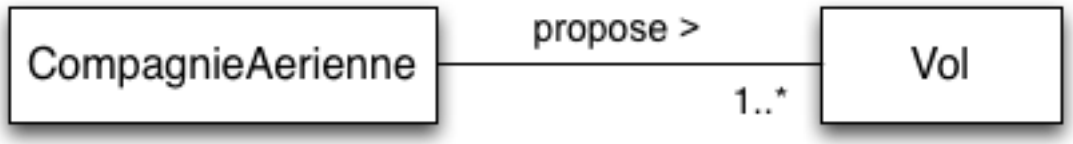
- Classes (propriétés, opérations)
- Associations (types, rôles, multiplicités)

Exemple (fil rouge)



À modéliser

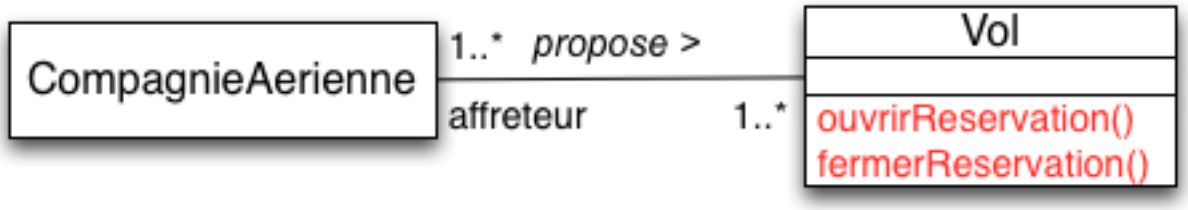
Des compagnies aériennes proposent différents vols



La **multiplicité** à une extrémité précise le nombre d'objets de la classe pouvant s'associer à un seul objet de la classe située à l'autre extrémité (1 , 0..1 , M..N , * , 1..*)

À modéliser

Un vol est ouvert/fermé à la réservation sur ordre de la compagnie

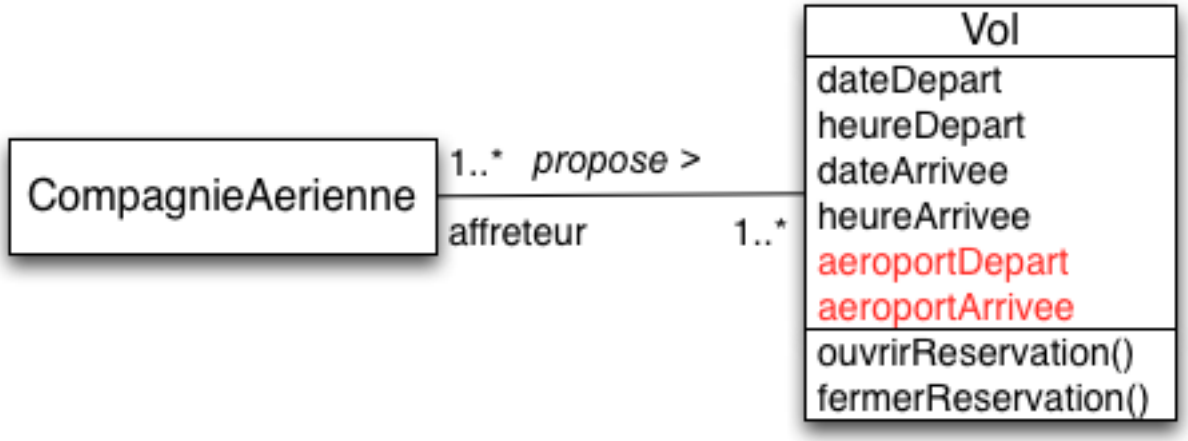


C'est l'objet sur lequel on pourra réaliser un traitement qui doit le déclarer en tant qu'opération. Les autres objets qui possèderont une référence dessus pourront alors lui envoyer un message qui invoque cette opération.

À modéliser

Un vol a un aéroport de départ et un aéroport d'arrivée

NE PAS FAIRE

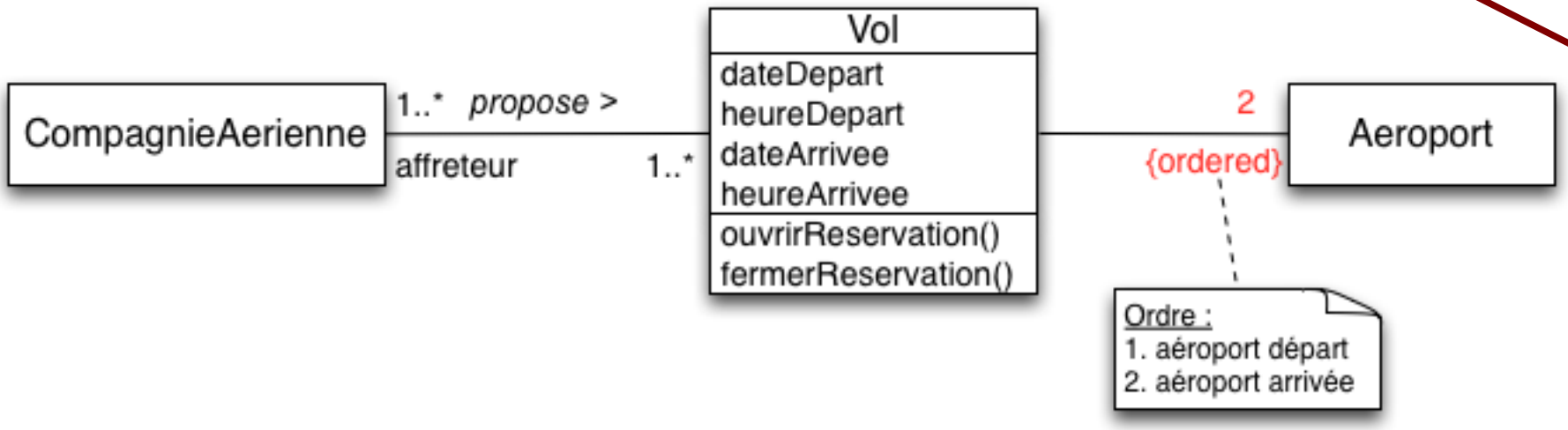


La notion d'aéroport est complexe.
Un aéroport ne possède pas seulement un nom...

À modéliser

Un vol a un aéroport de départ et un aéroport d'arrivée

NE PAS FAIRE

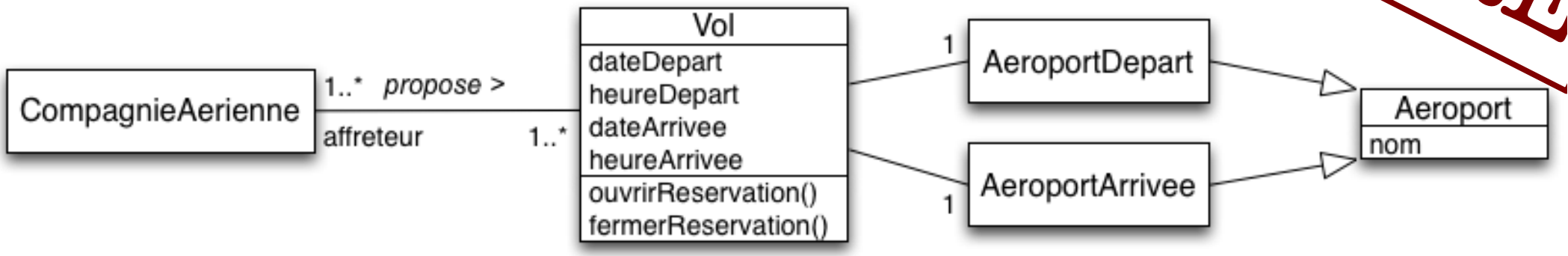


Modélisation peu explicite (pour l'expert métier).
Solution purement technique, incorrecte conceptuellement.

À modéliser

Un vol a un aéroport de départ et un aéroport d'arrivée

NE PAS FAIRE

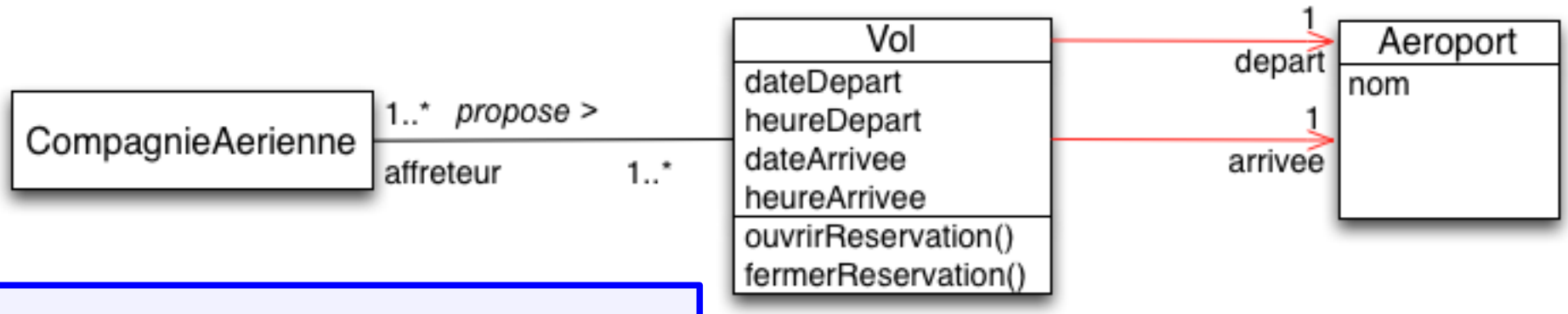


Un aeroport n'est pas *aeroport de départ* ou *aeroport d'arrivée* indépendamment d'un vol.

À modéliser

Un vol a un aéroport de départ et un aéroport d'arrivée

FAIRE

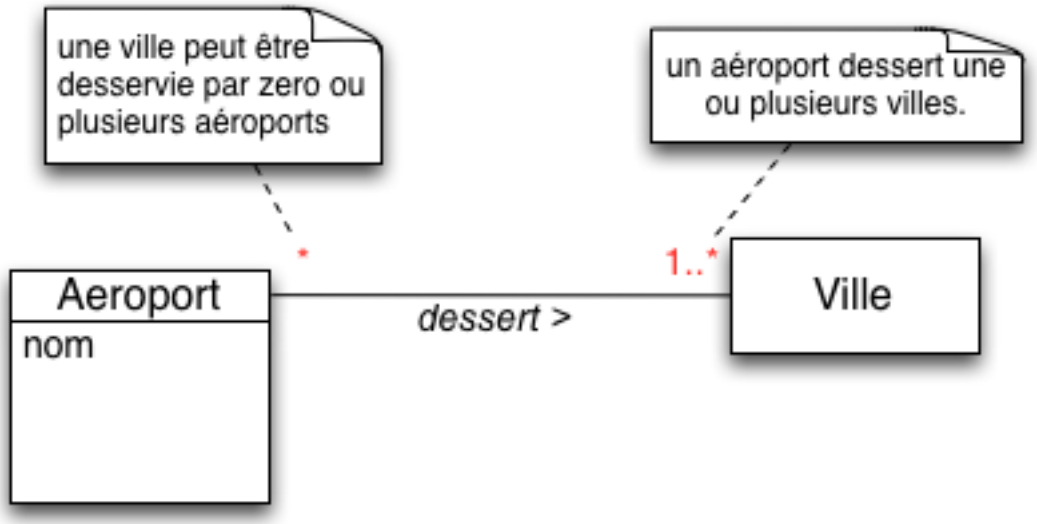



```
public class Vol {

    private Date dateHDepart;
    private Date dateHArrivee;
    private Aeroport depart;
    private Aeroport arrivee;
    ...
}
```

```
public class Aeroport {

    private String nom;
    ...
}
```

À modéliser

Chaque aéroport dessert une ou plusieurs villes

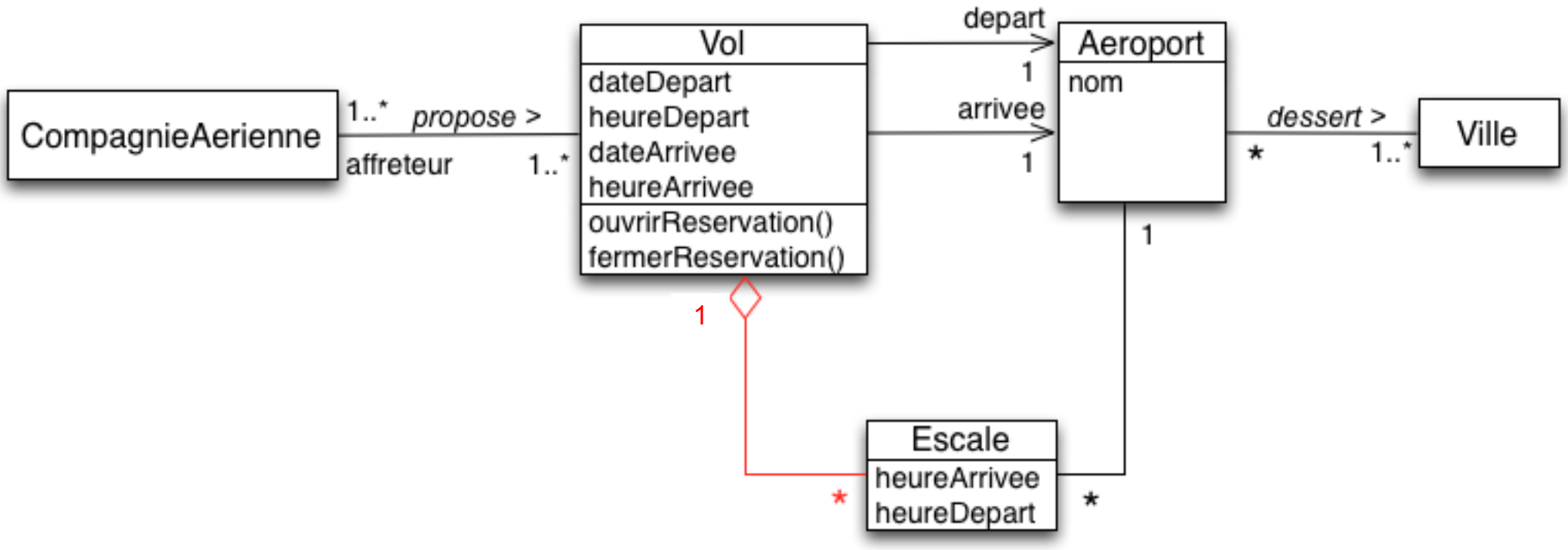



```
public class Aeroport {

    private String nom;
    private Ville[] villes;
    ...
}
```

```
public class Ville {

    private Aeroport[] aeroports;
    ...
}
```

À modéliser

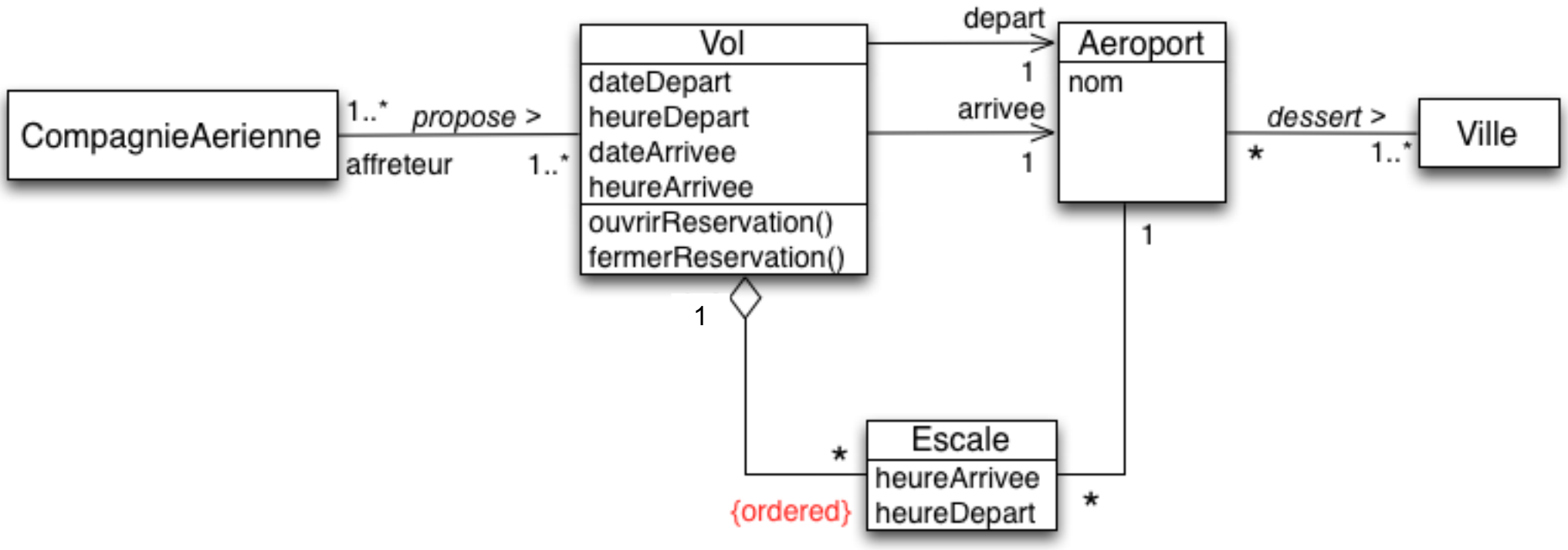
Un vol peut comporter des escales dans des aéroports.
Une escale a une heure d'arrivée et une heure de départ.



Une **agrégation** modélise une inclusion qui ne contraint pas : la navigabilité, les multiplicités, la durée de vie des éléments agrégés.

Notons

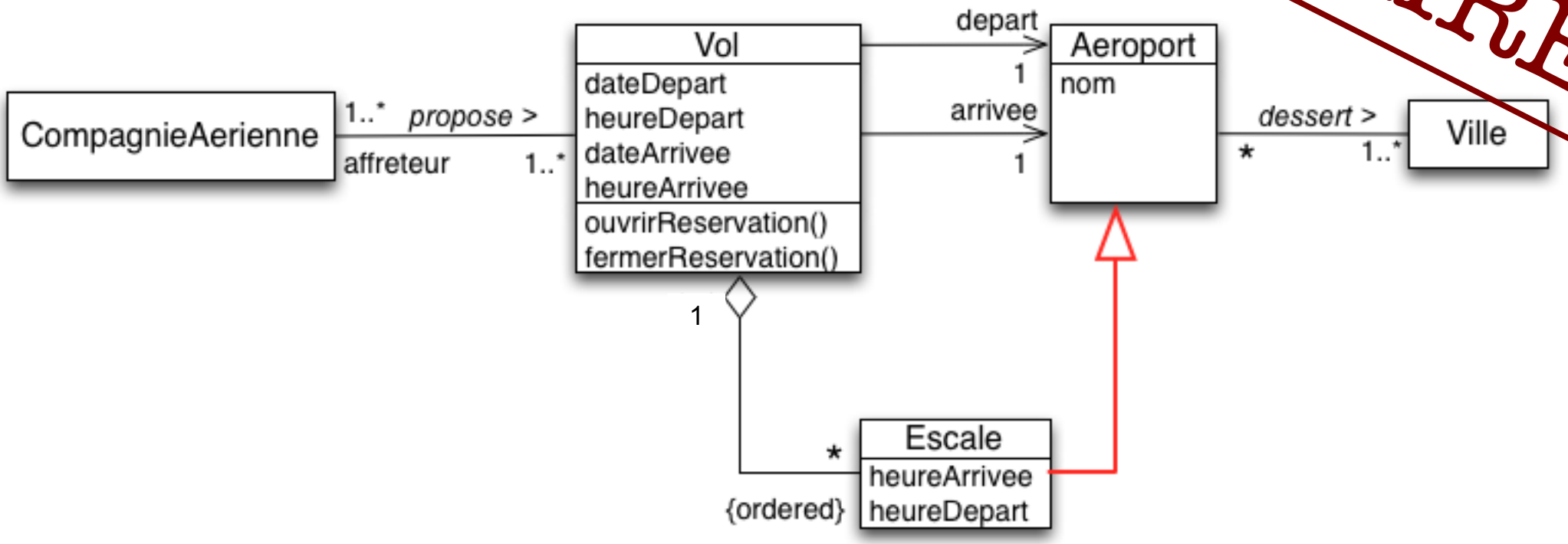
La classe *Escale* comporte peu d'informations propres ; elle est forcément associée à **un** aéroport et n'existe pas par elle-même.



Notons

La classe *Escale* comporte peu d'informations propres ; elle est forcément associée à **un** aéroport et n'existe pas par elle-même.

NE PAS FAIRE

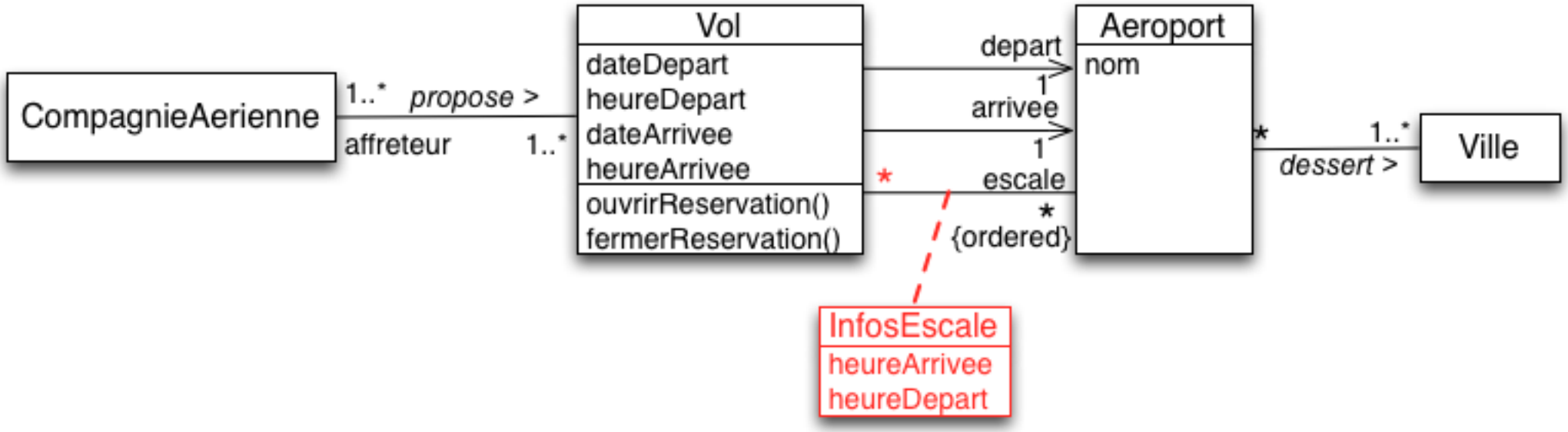


Il s'agit ici d'un héritage d'implémentation qui est fortement déconseillé pour permettre une évolution cohérente du modèle.

Notons

La classe *Escale* comporte peu d'informations propres ; elle est forcément associée à **un** aéroport et n'existe pas par elle-même.

FAIRE

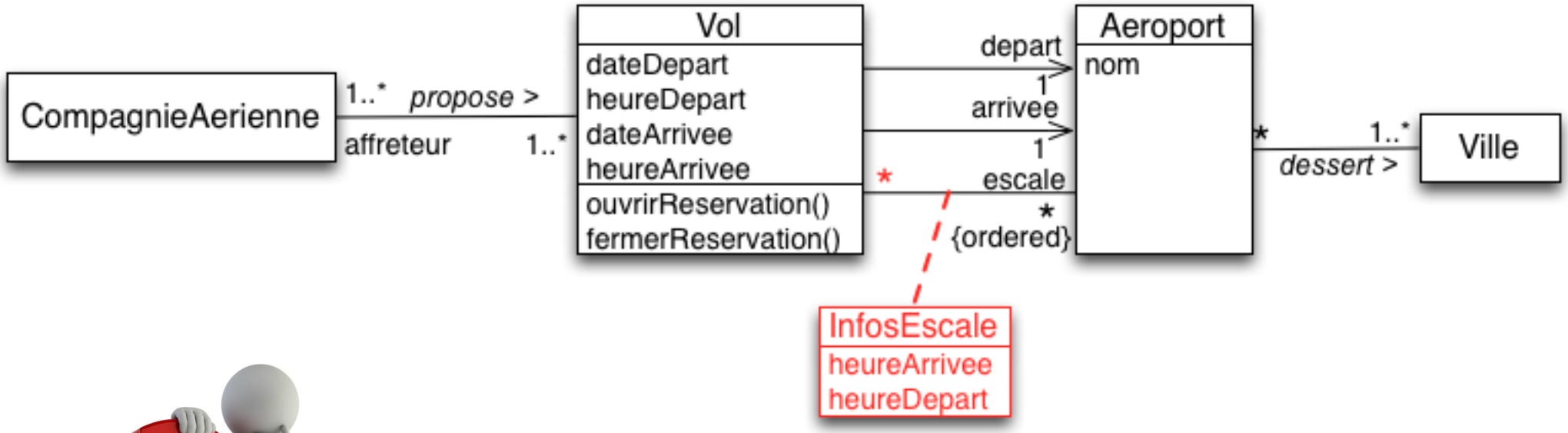


Une **classe association** permet d'ajouter des propriétés et des opérations à une association.

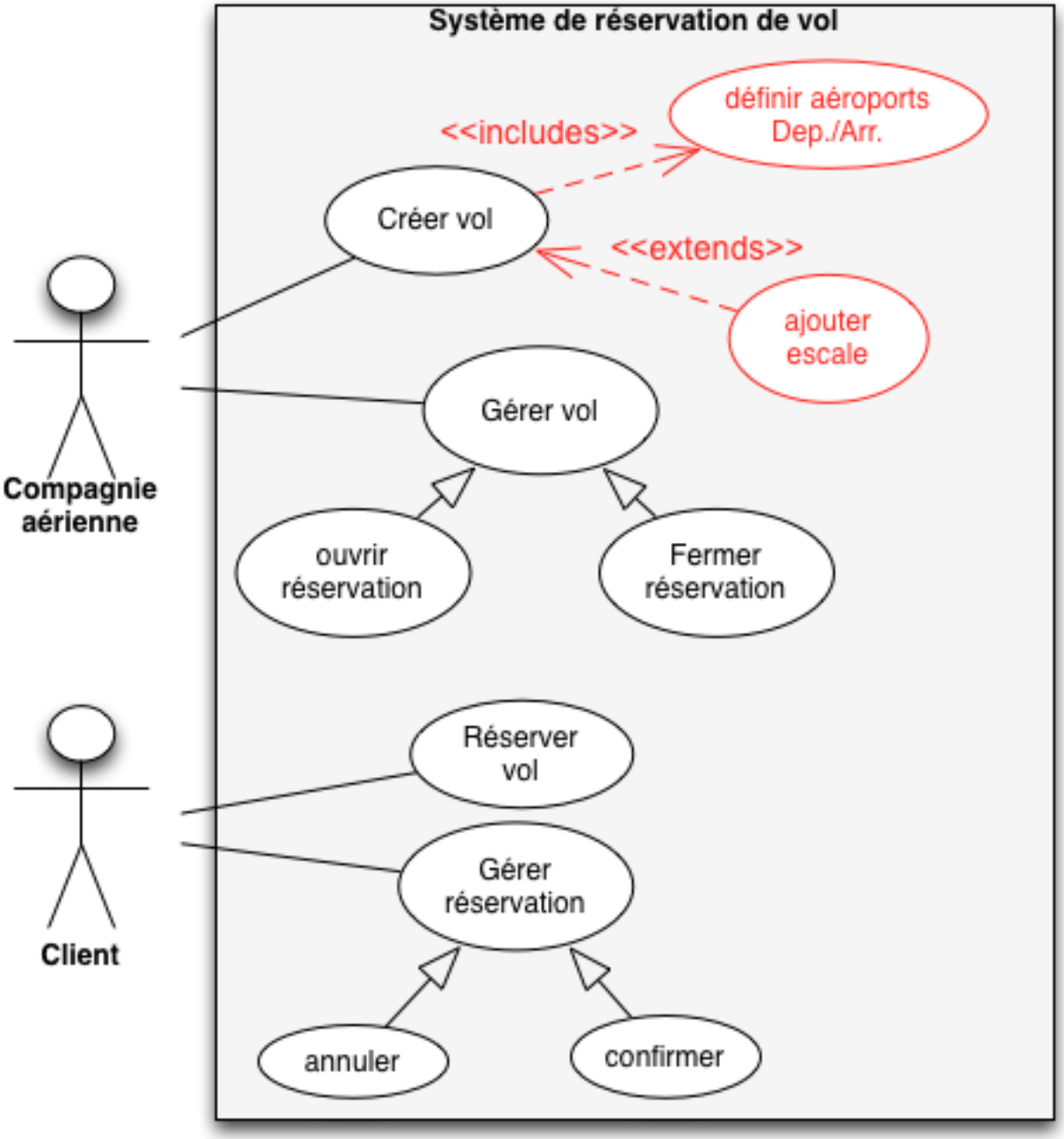
Notons

La classe *Escale* comporte peu d'informations propres ; elle est forcément associée à **un** aéroport et n'existe pas par elle-même.

FAIRE

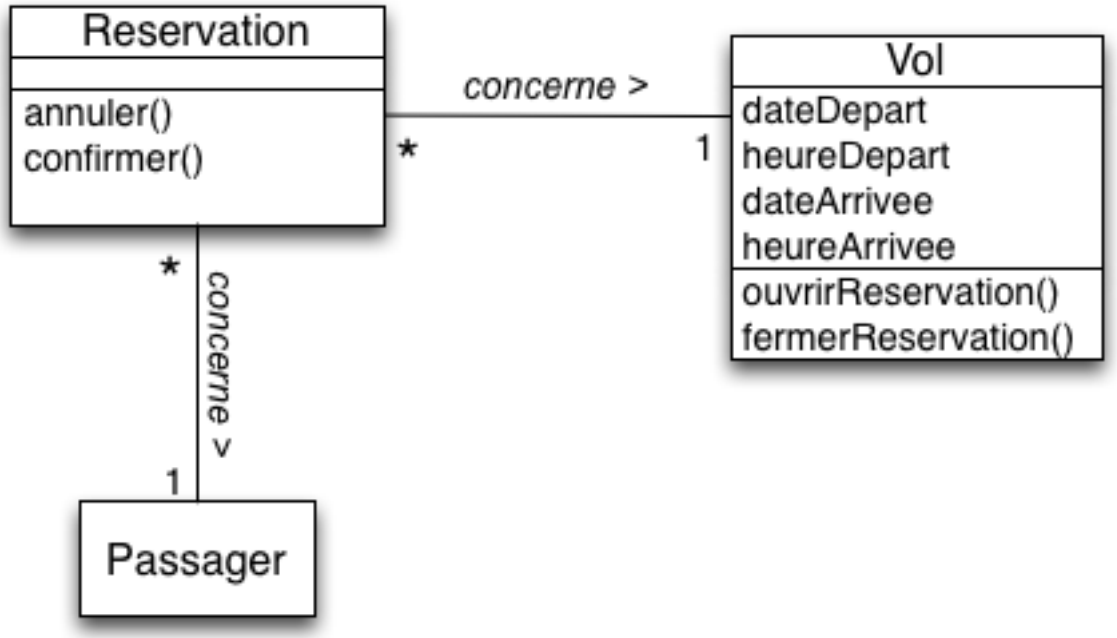


Comment implémenter une « classe association » ?


Système de réservation de vol
définir aéroports Dep./Arr.
<<includes>>
Créer vol
<<extends>>
ajouter escale
Gérer vol
ouvrir réservation
Fermer réservation
Compagnie aérienne
Réserver vol
Gérer réservation
annuler
confirmer
Client

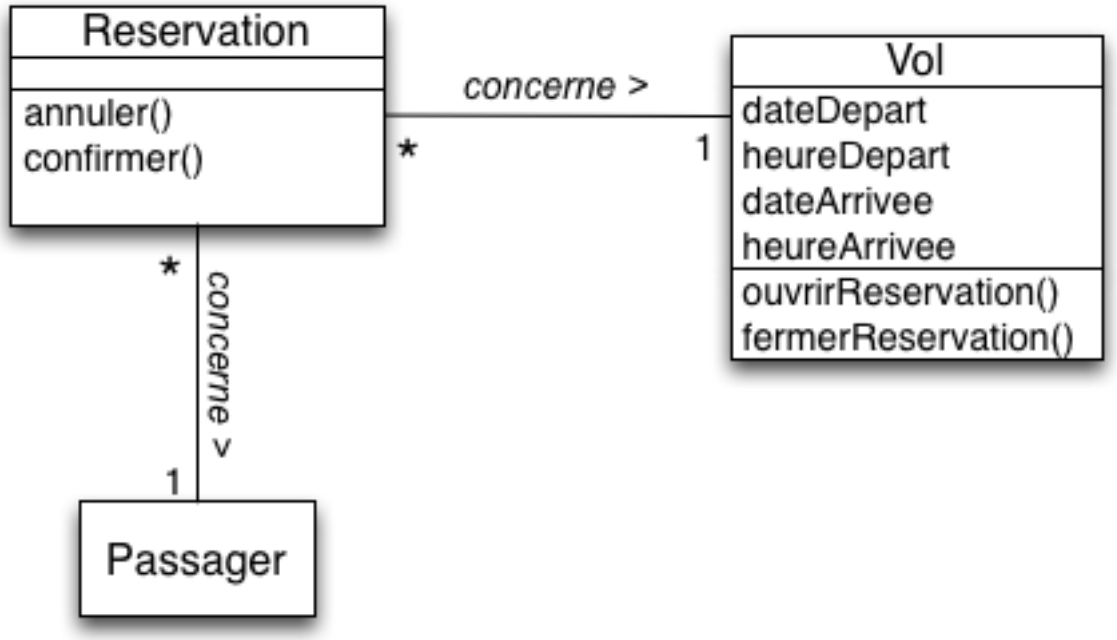
À modéliser

Un client peut réserver un ou plusieurs vols, pour des passagers différents.
Une réservation concerne un seul vol et un seul passager.
Une réservation peut être annulée ou confirmée.



À modéliser

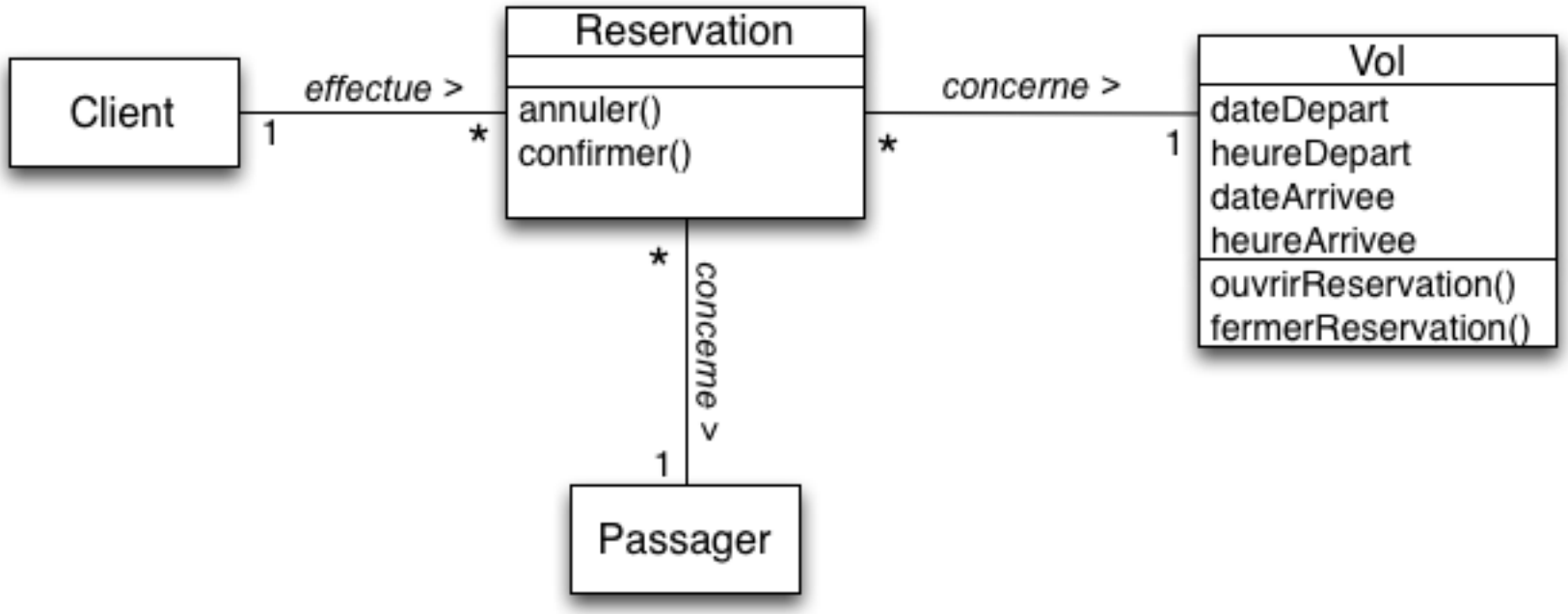
Un client peut réserver un ou plusieurs vols, pour des passagers différents.
Une réservation concerne un seul vol et un seul passager.
Une réservation peut être annulée ou confirmée.



Placer le client...

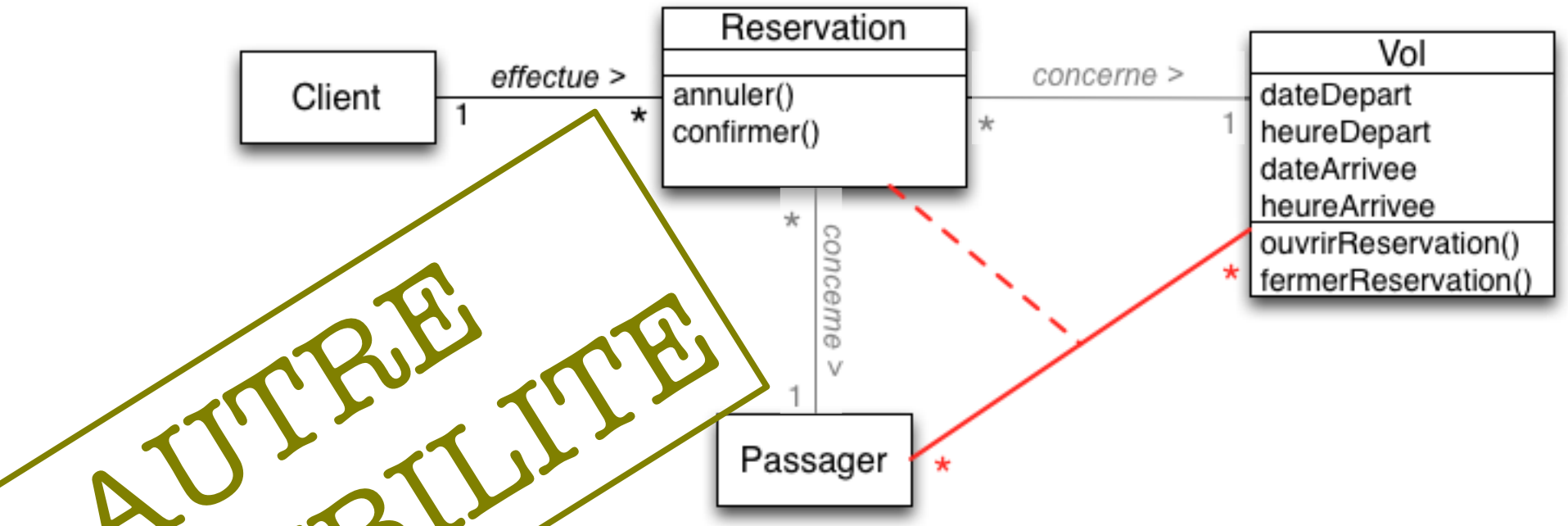
À modéliser

Un client peut réserver un ou plusieurs vols, pour des passagers différents.
Une réservation concerne un seul vol et un seul passager.
Une réservation peut être annulée ou confirmée.

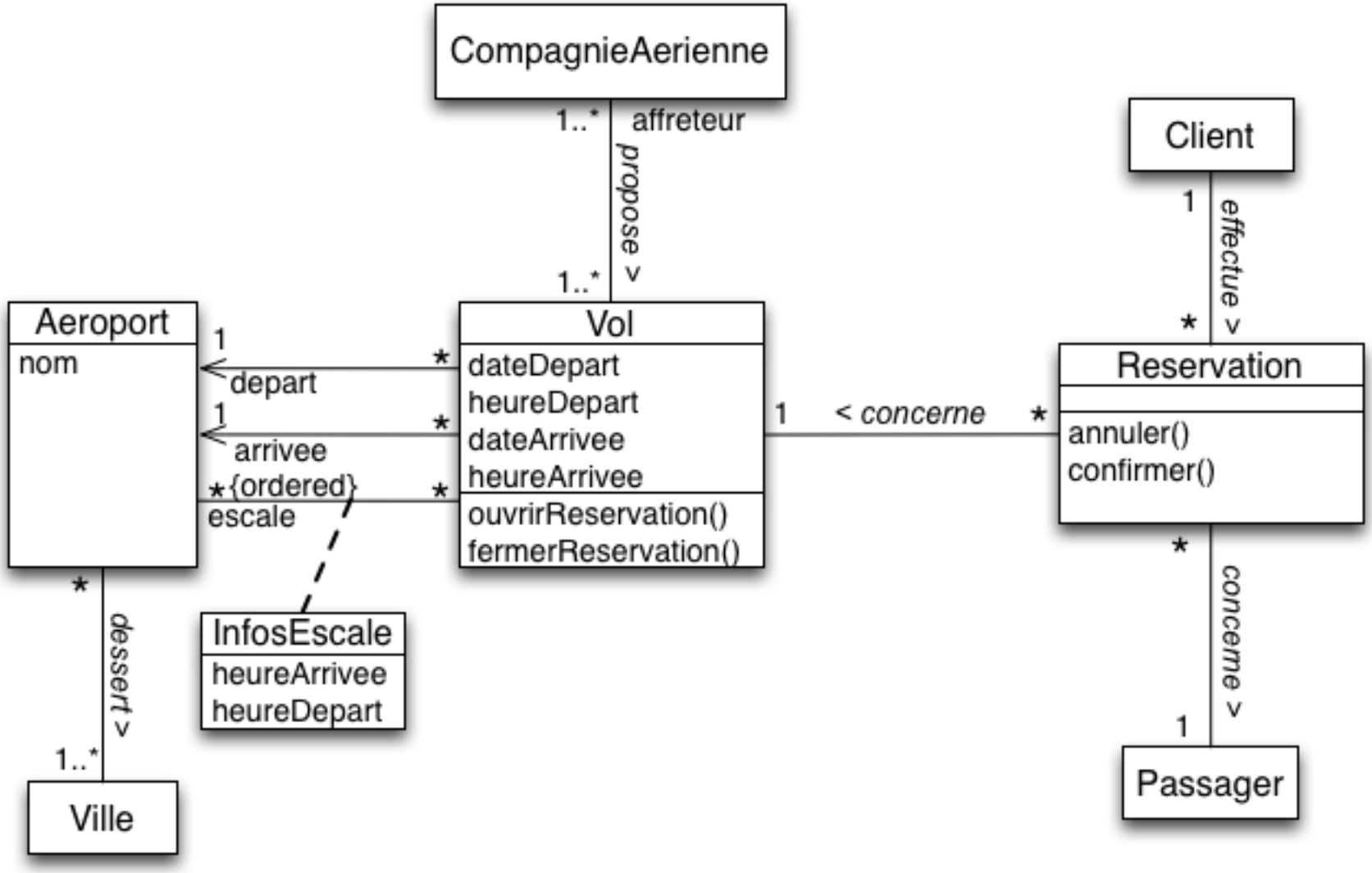


À modéliser

Un client peut réserver un ou plusieurs vols, pour des passagers différents.
Une réservation concerne un seul vol et un seul passager.
Une réservation peut être annulée ou confirmée.

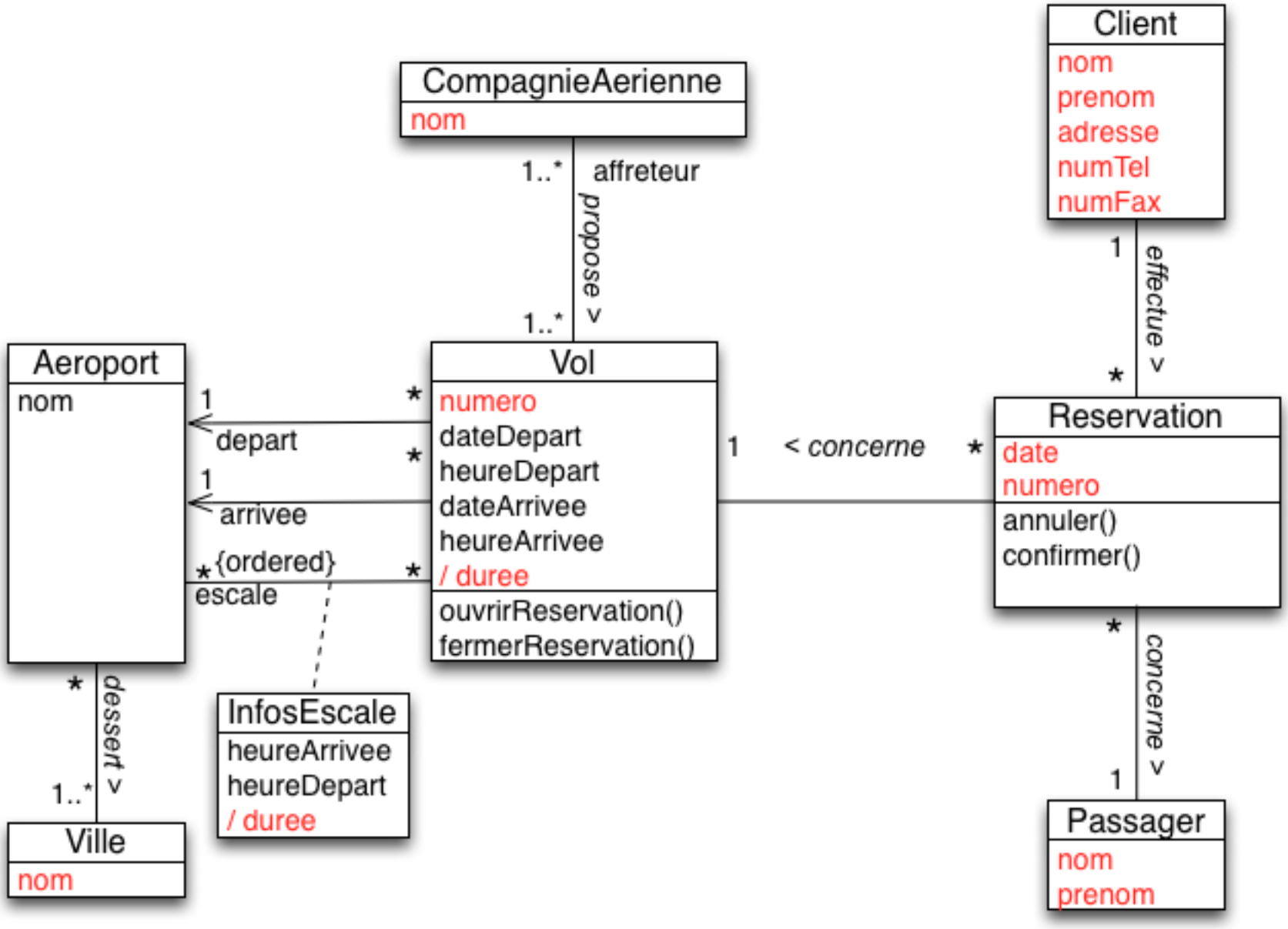


Bilan (provisoire)



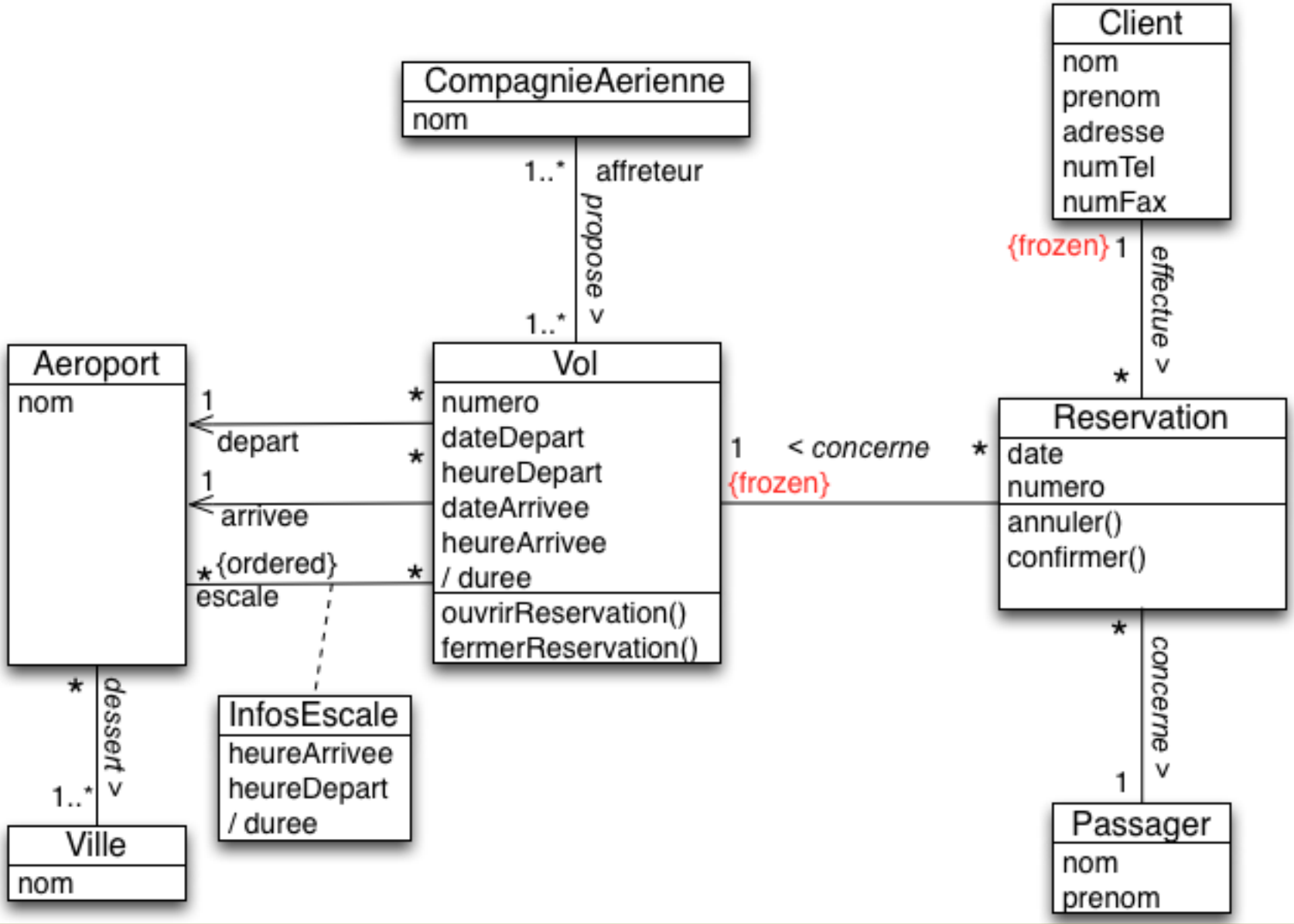
À modéliser

Ajout d'attributs métier et dérivés



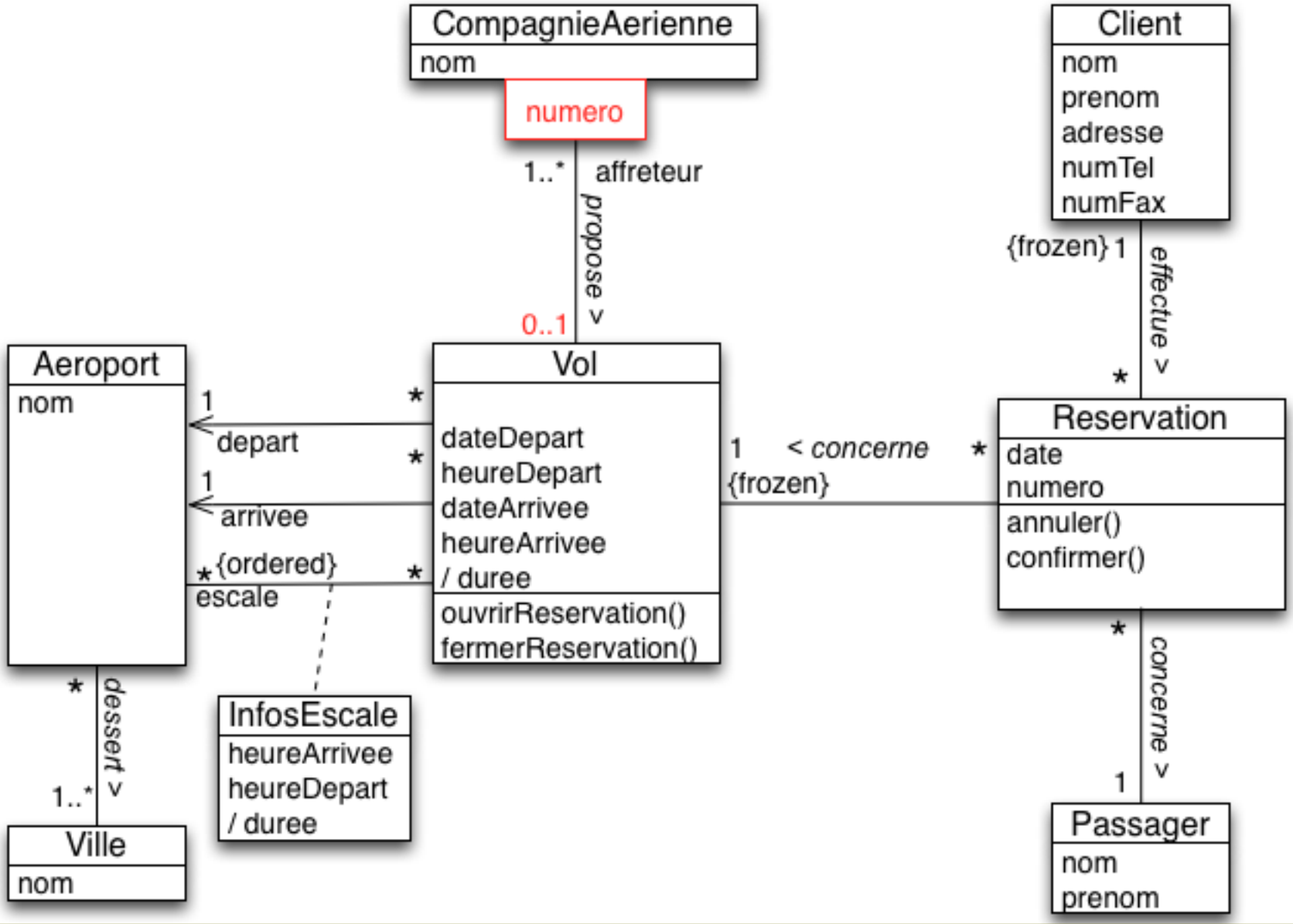
À modéliser

Modifier le vol (ou le client) d'une réservation nécessite de supprimer la réservation et d'en créer une nouvelle.



À modéliser

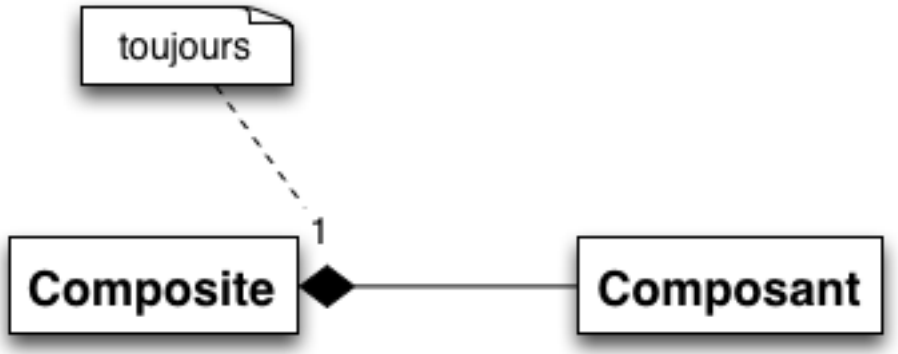
Chaque vol est identifié de façon unique par un numéro propre à la compagnie.



Compléments sur ...

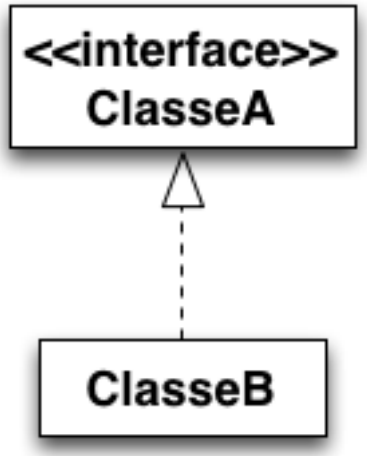
… l'association qualifiée

… la composition

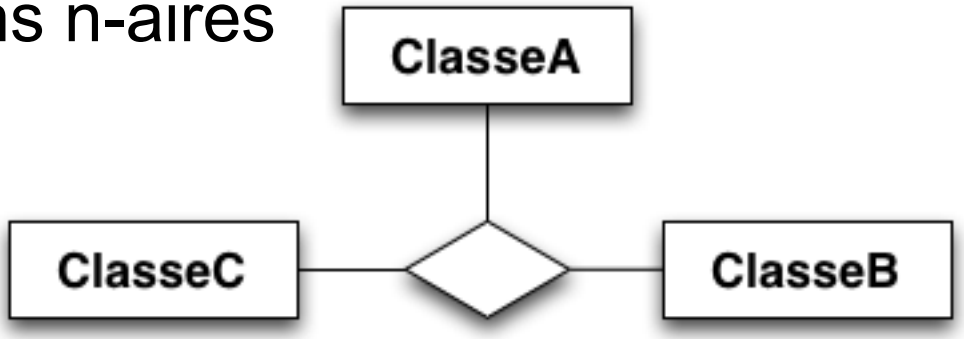


Compléments sur ...

… les interfaces

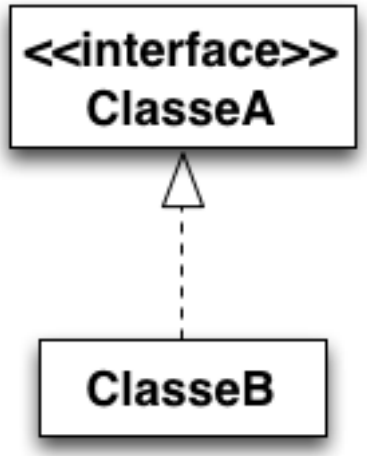


… les associations n-aires

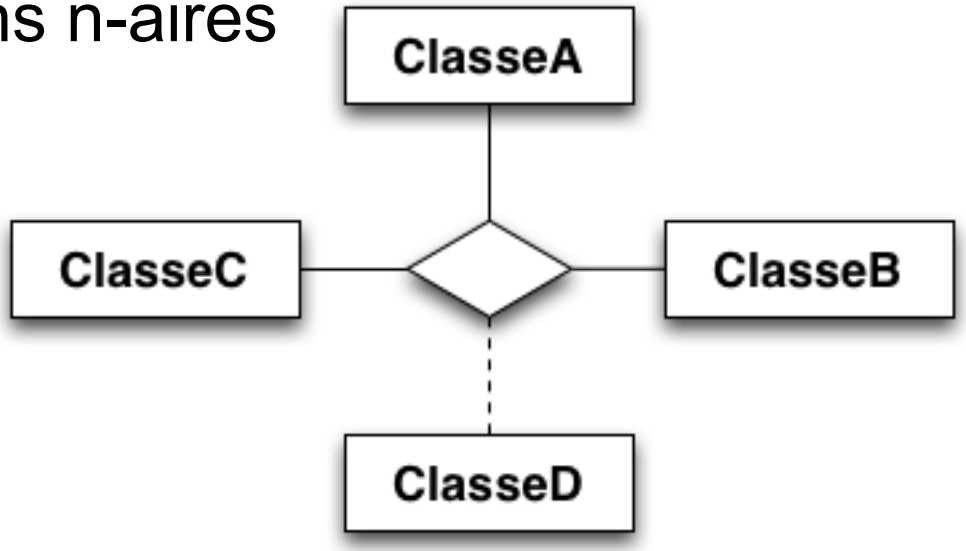


Compléments sur ...

… les interfaces



… les associations n-aires



Quelques règles de correspondance Java vers UML

1. à toute classe Java doit correspondre une classe UML (de même nom)

2. à toute interface Java doit correspondre une interface UML (de même nom)

3. À tout attribut de type primitif d'une classe Java doit correspondre une propriété (de même nom)

3.bis à tout attribut dont le type est une autre classe Java doit correspondre une association UML.

4. Si une classe Java hérite d'une autre classe Java, les classes UML correspondantes doivent avoir elles aussi une relation d'héritage.

5. Si une classe Java réalise une interface, la classe UML correspondante doit aussi réaliser l'interface UML correspondante.