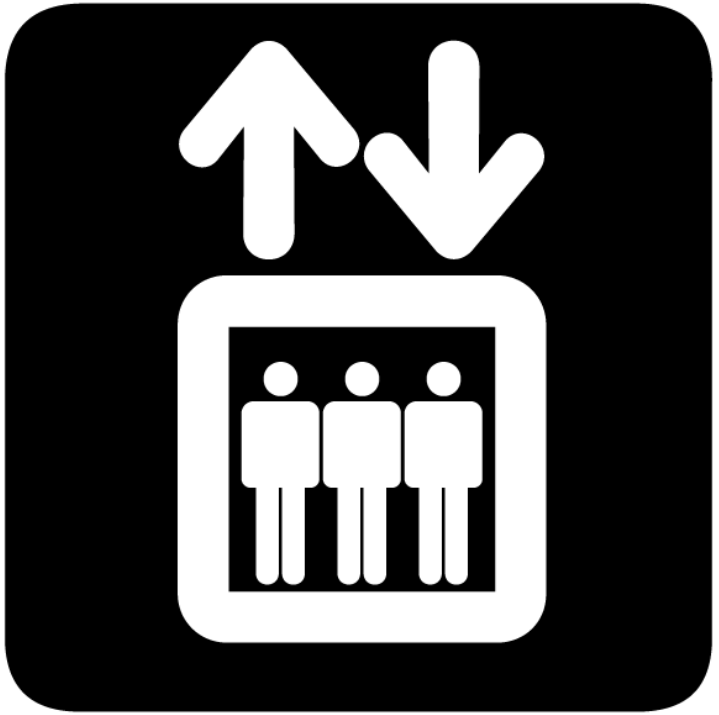
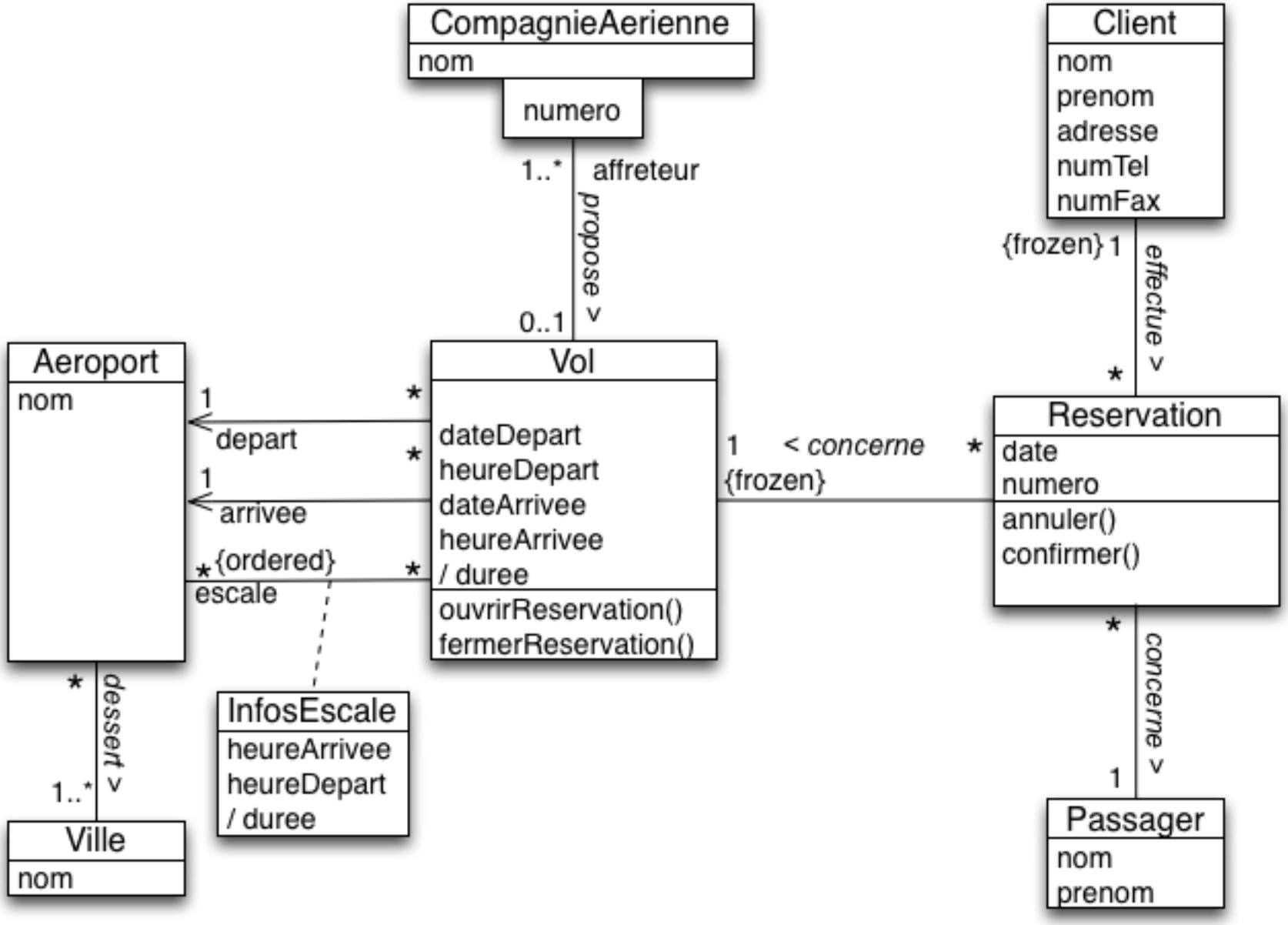
Diagramme de classes

Raisonnement et abstraction

Au programme



Retour sur le SI Vols



Principe de « forte cohésion »

- La cohésion mesure la compréhensibilité des classes

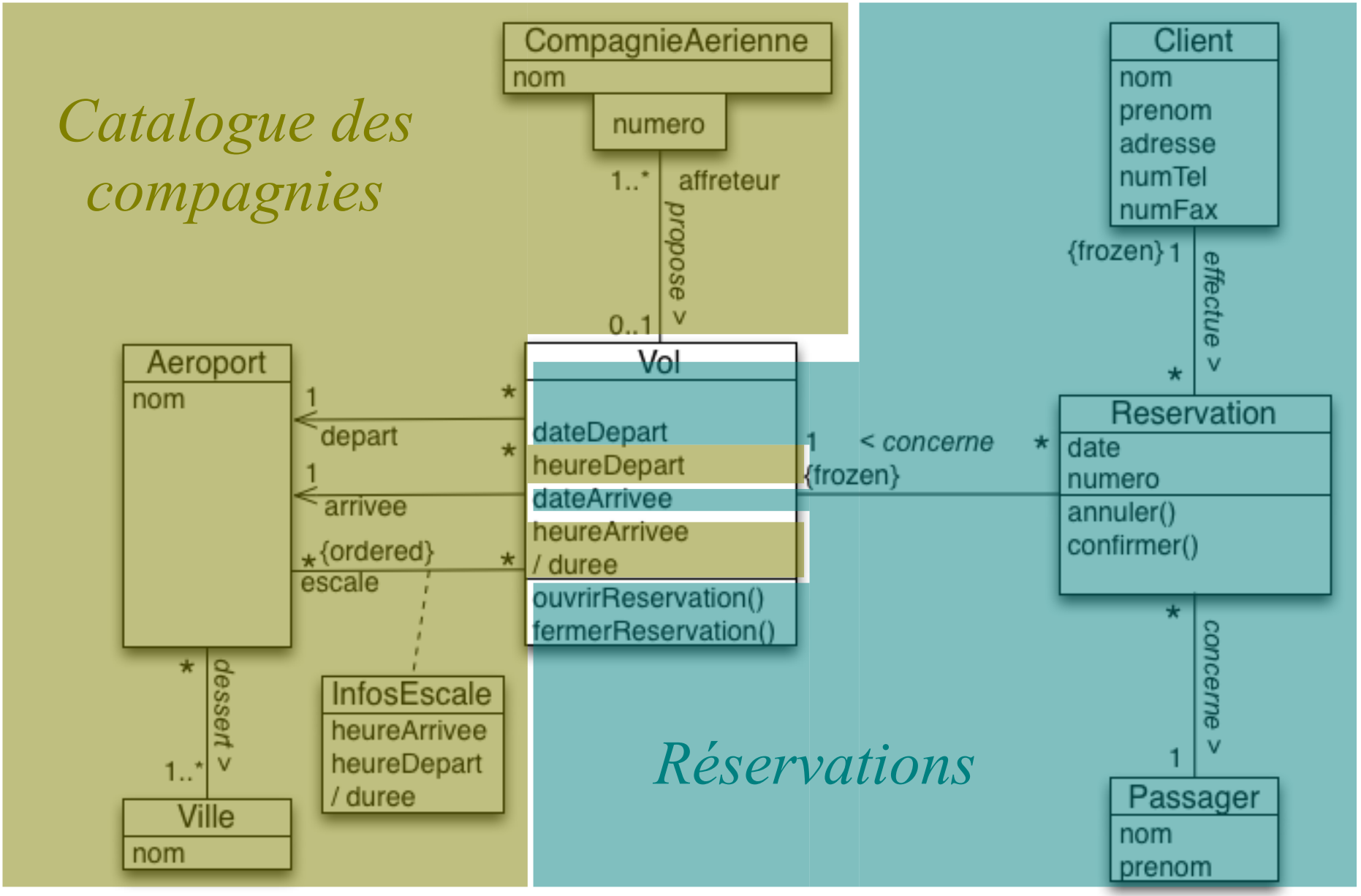
« une classe doit avoir des responsabilités cohérentes, et ne doit pas avoir des responsabilités trop variées. »

Ceci afin d'assurer une meilleure :

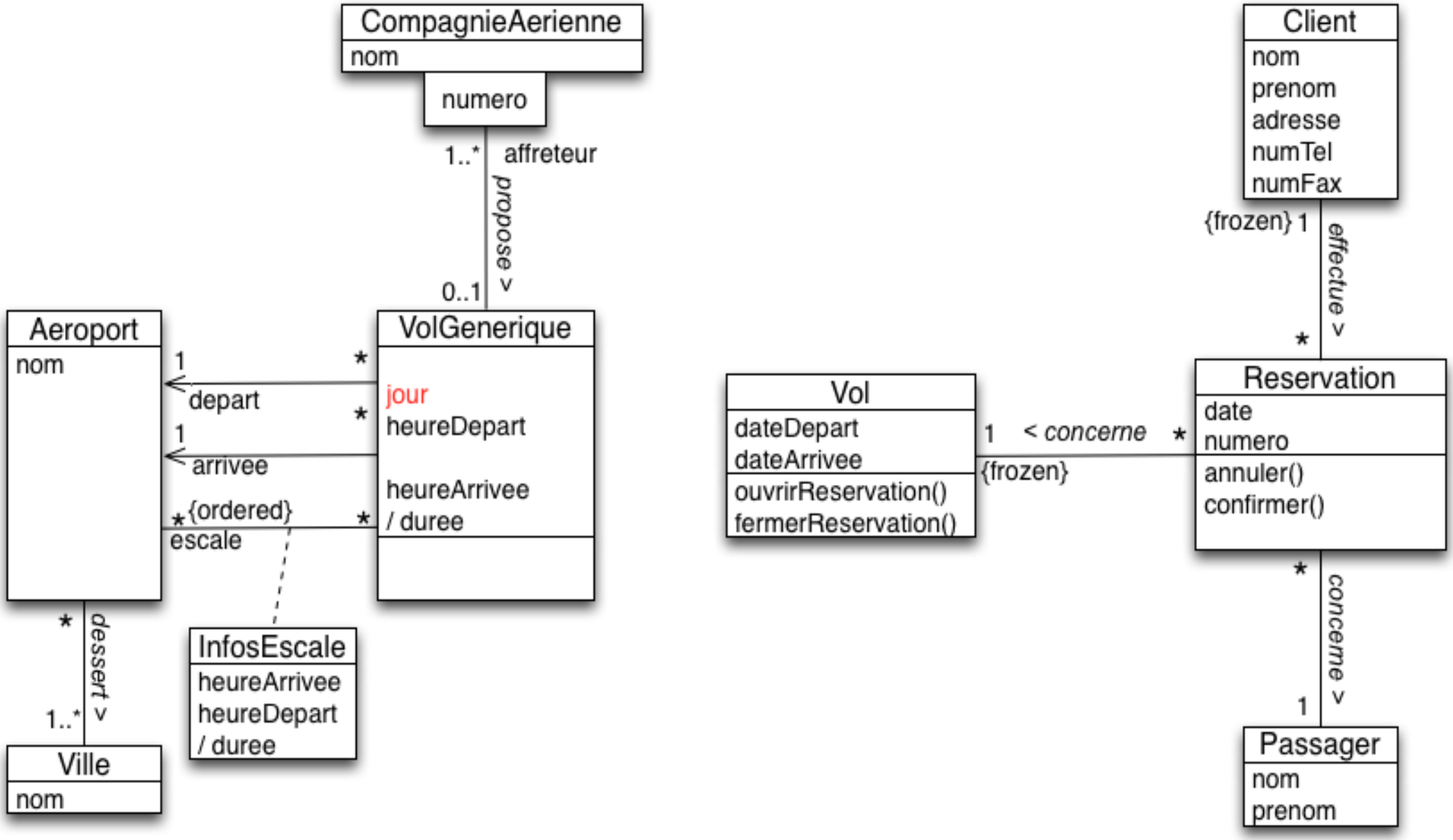
- compréhension
- maintenance
- réutilisation

} de la classe

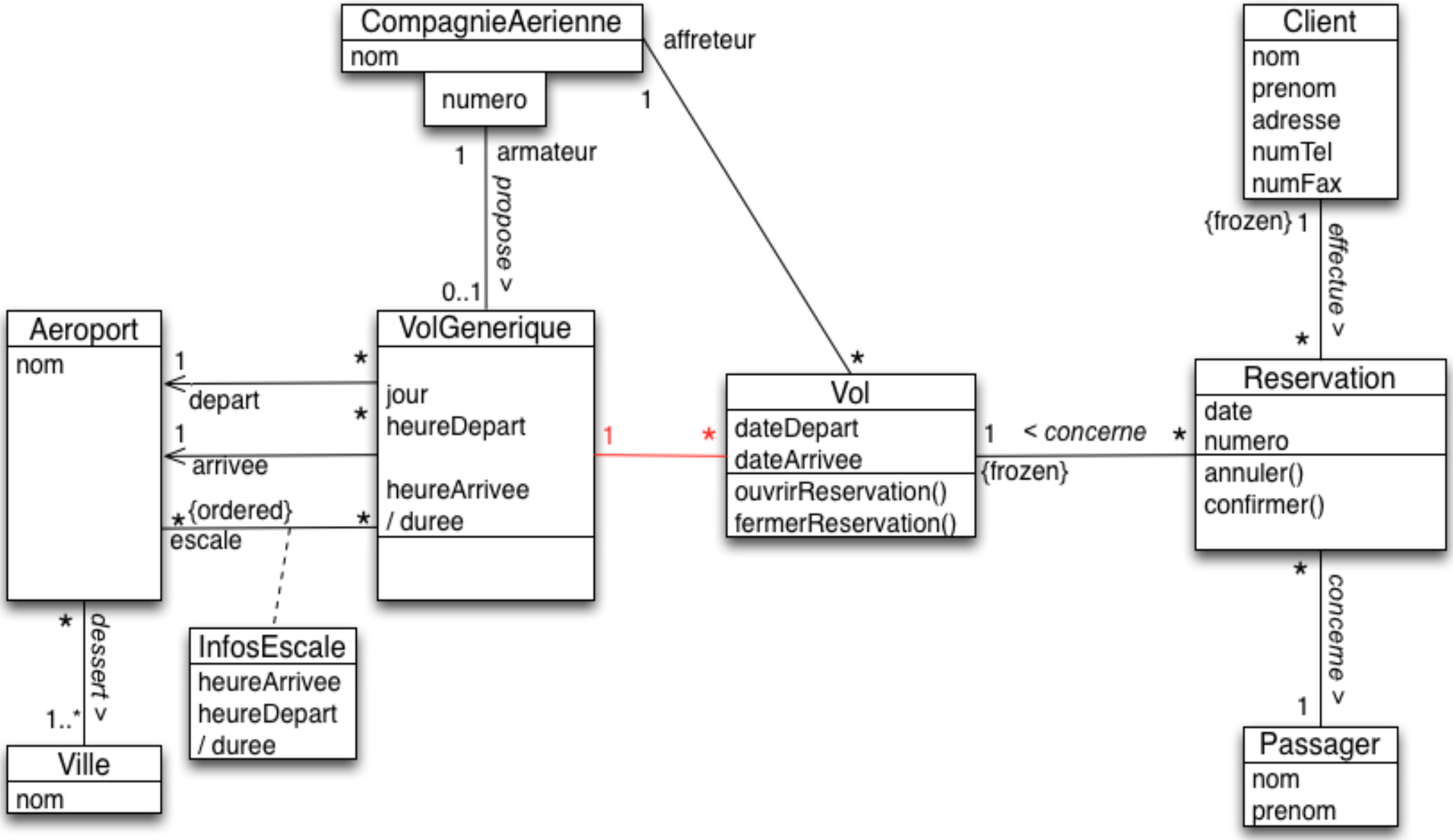
Principe de forte cohésion : non respecté



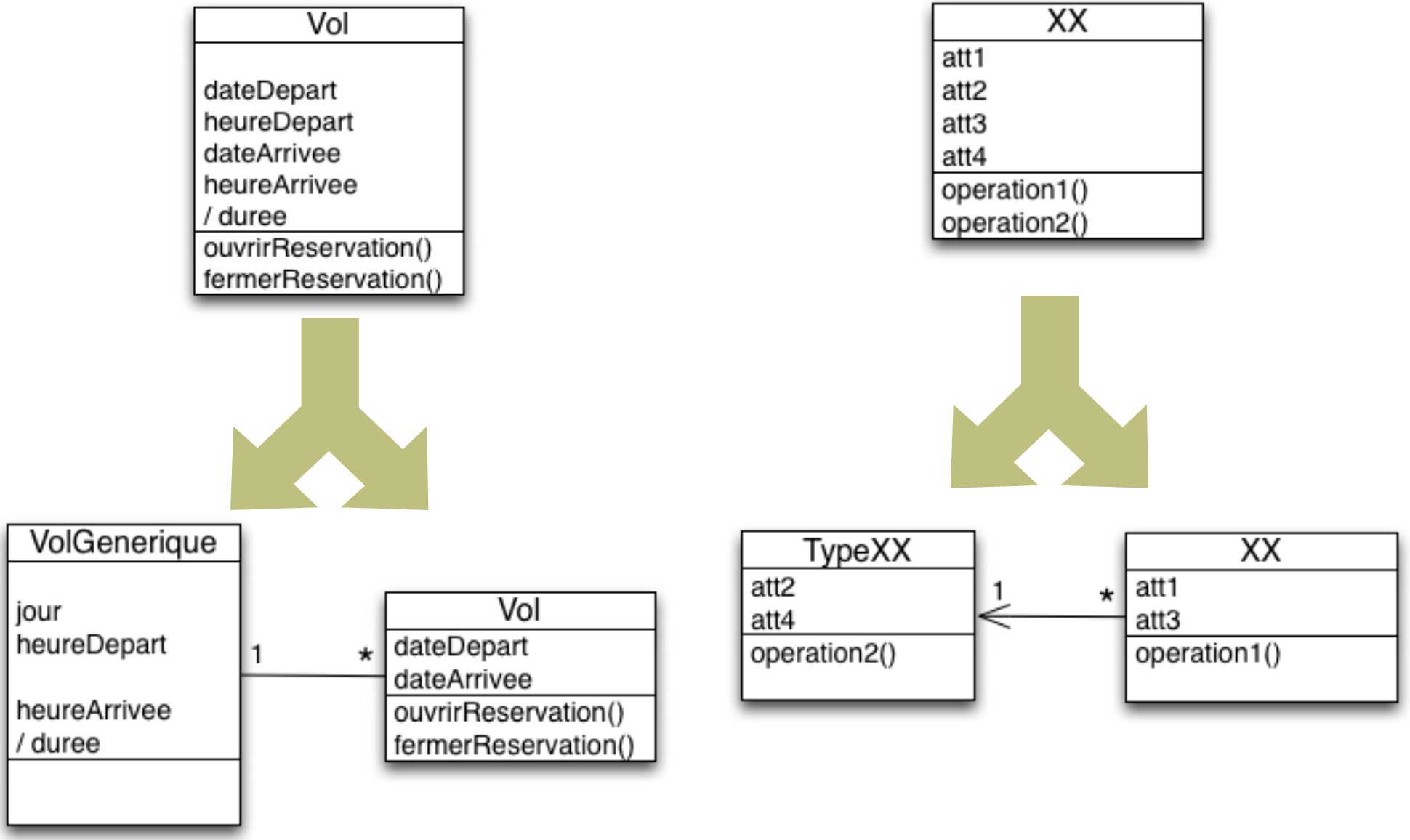
Principe de forte cohésion



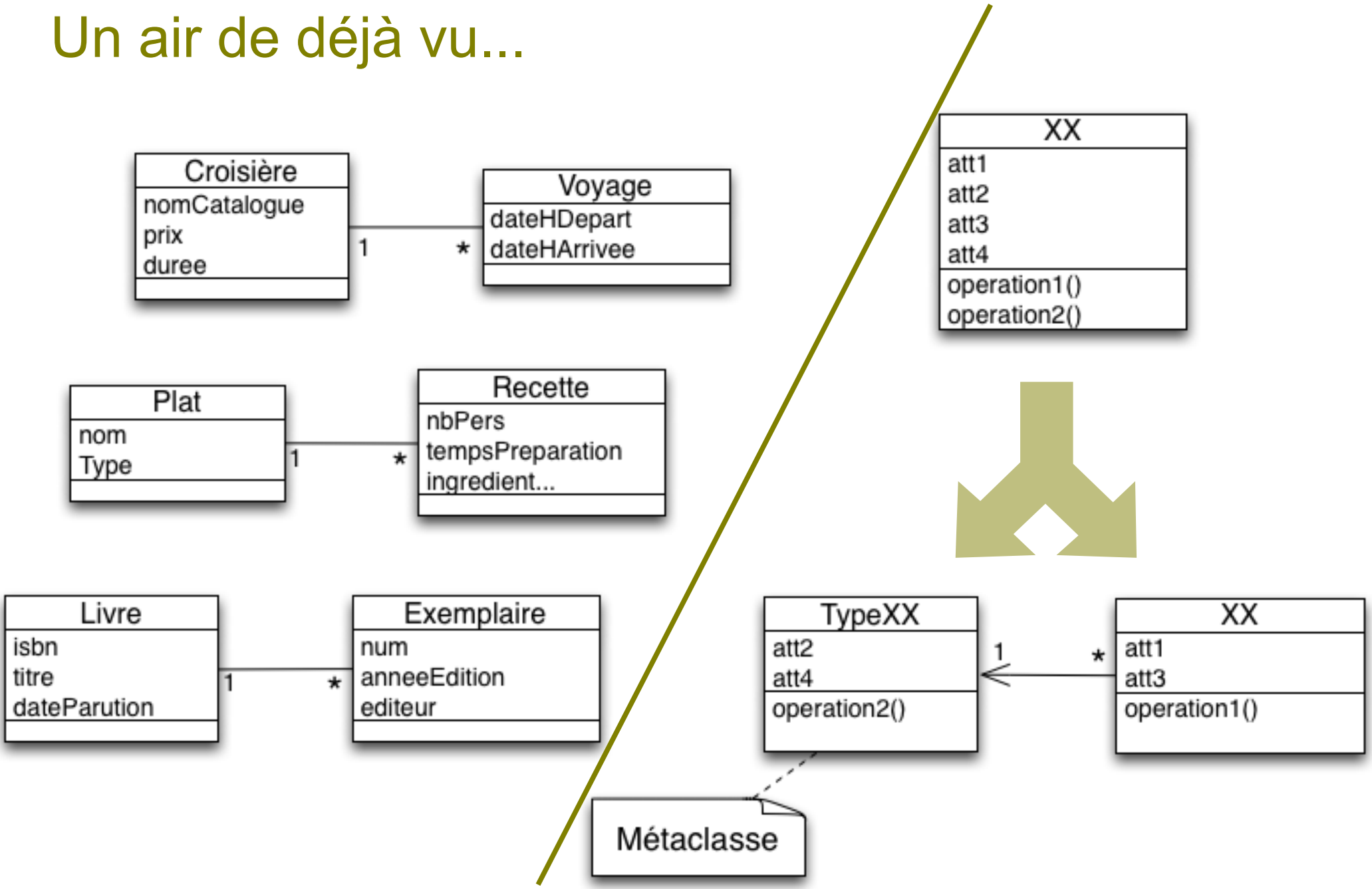
Principe de forte cohésion



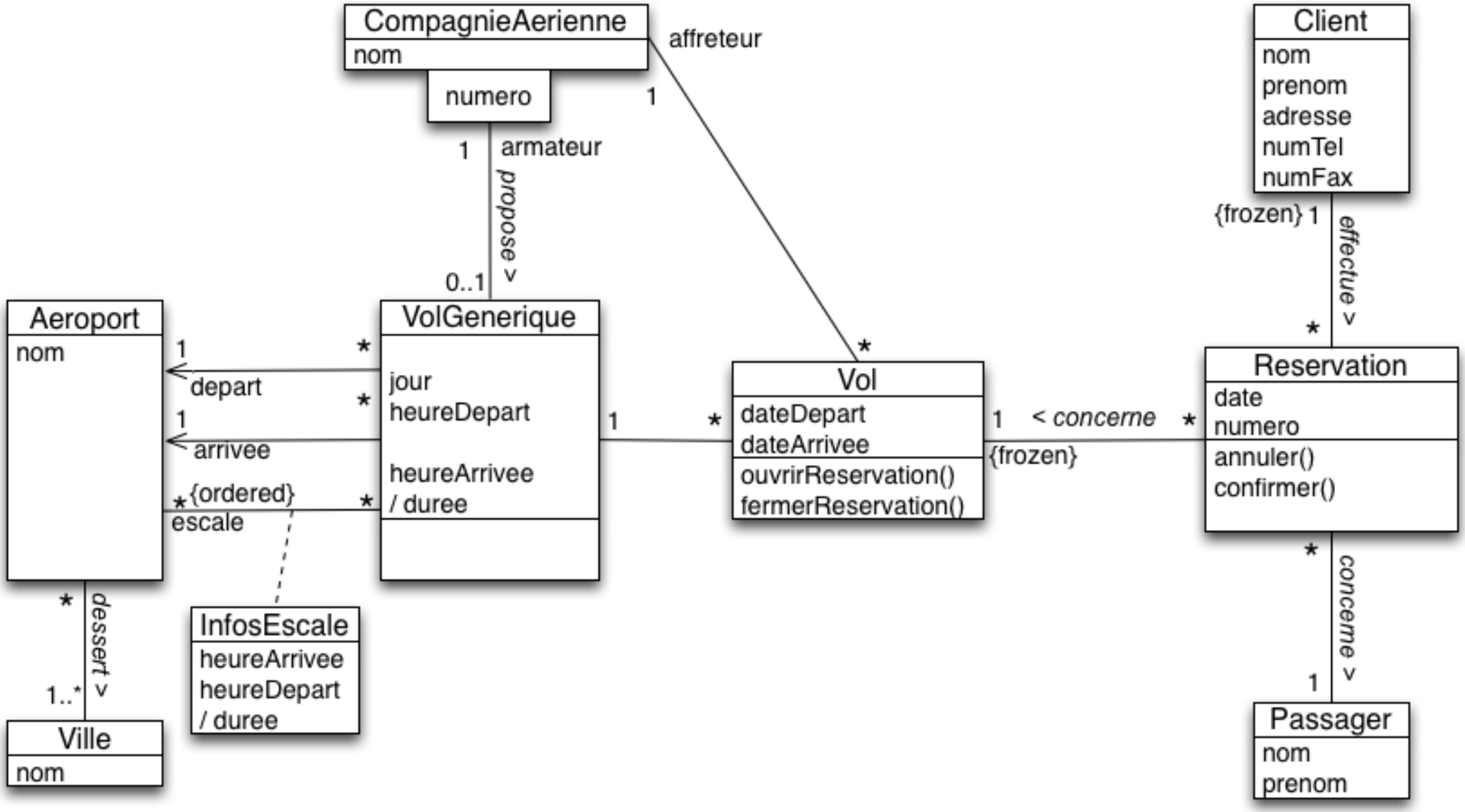
Pattern de la « métaclasse »



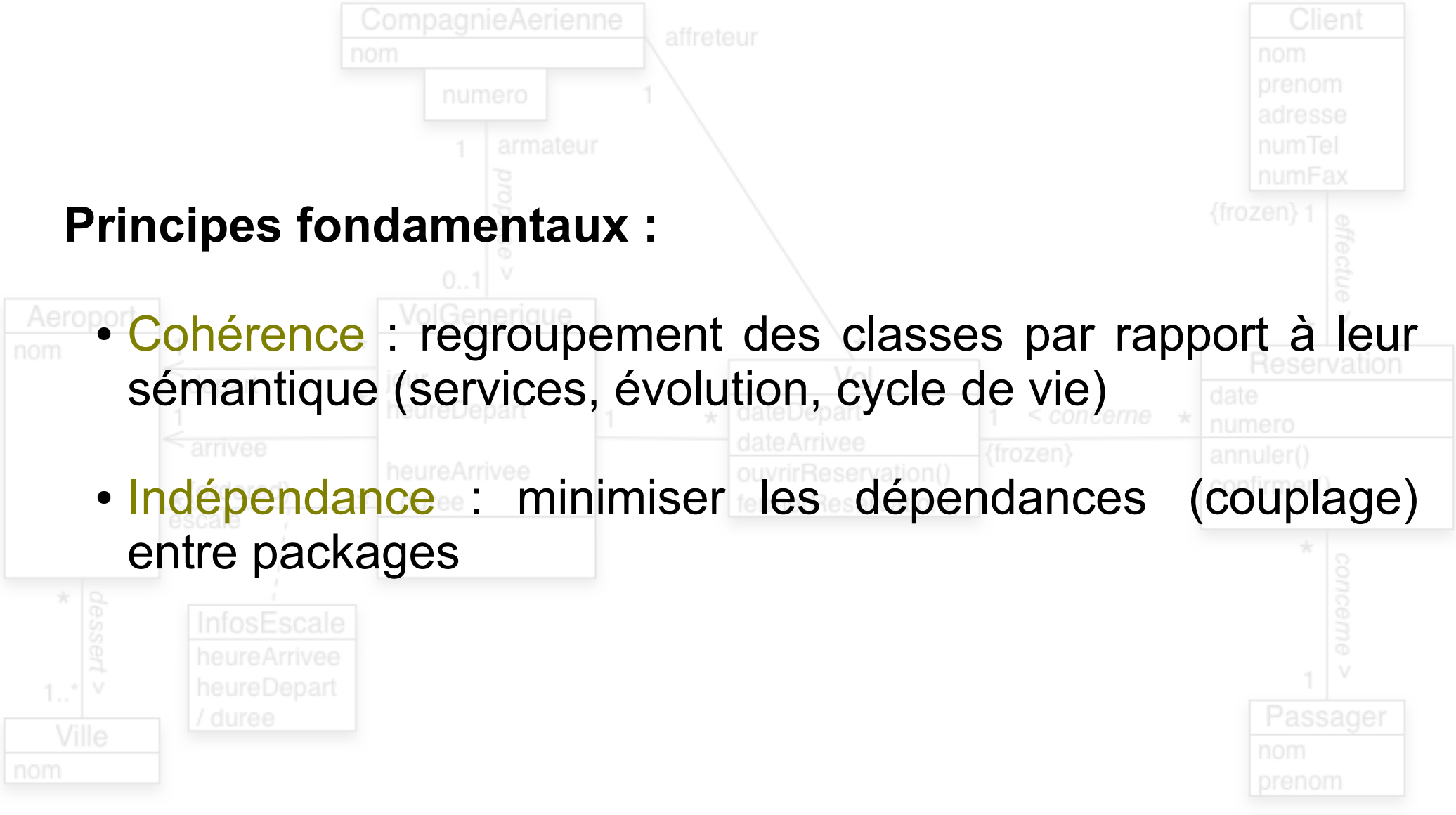
Un air de déjà vu...



Structuration du modèle statique



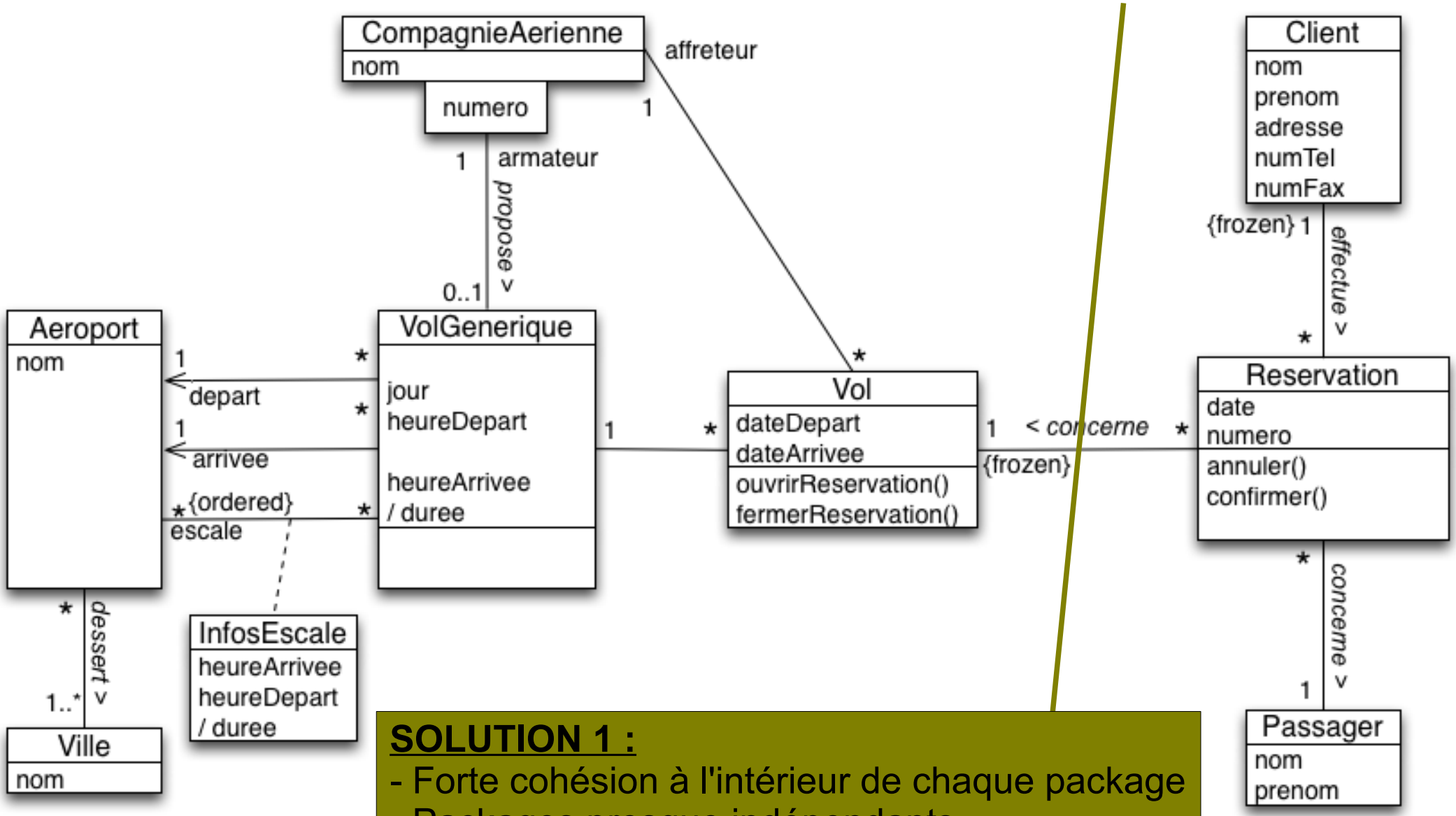
Structuration du modèle statique (packages)



Principes fondamentaux :

- Cohérence : regroupement des classes par rapport à leur sémantique (services, évolution, cycle de vie)
- Indépendance : minimiser les dépendances (couplage) entre packages

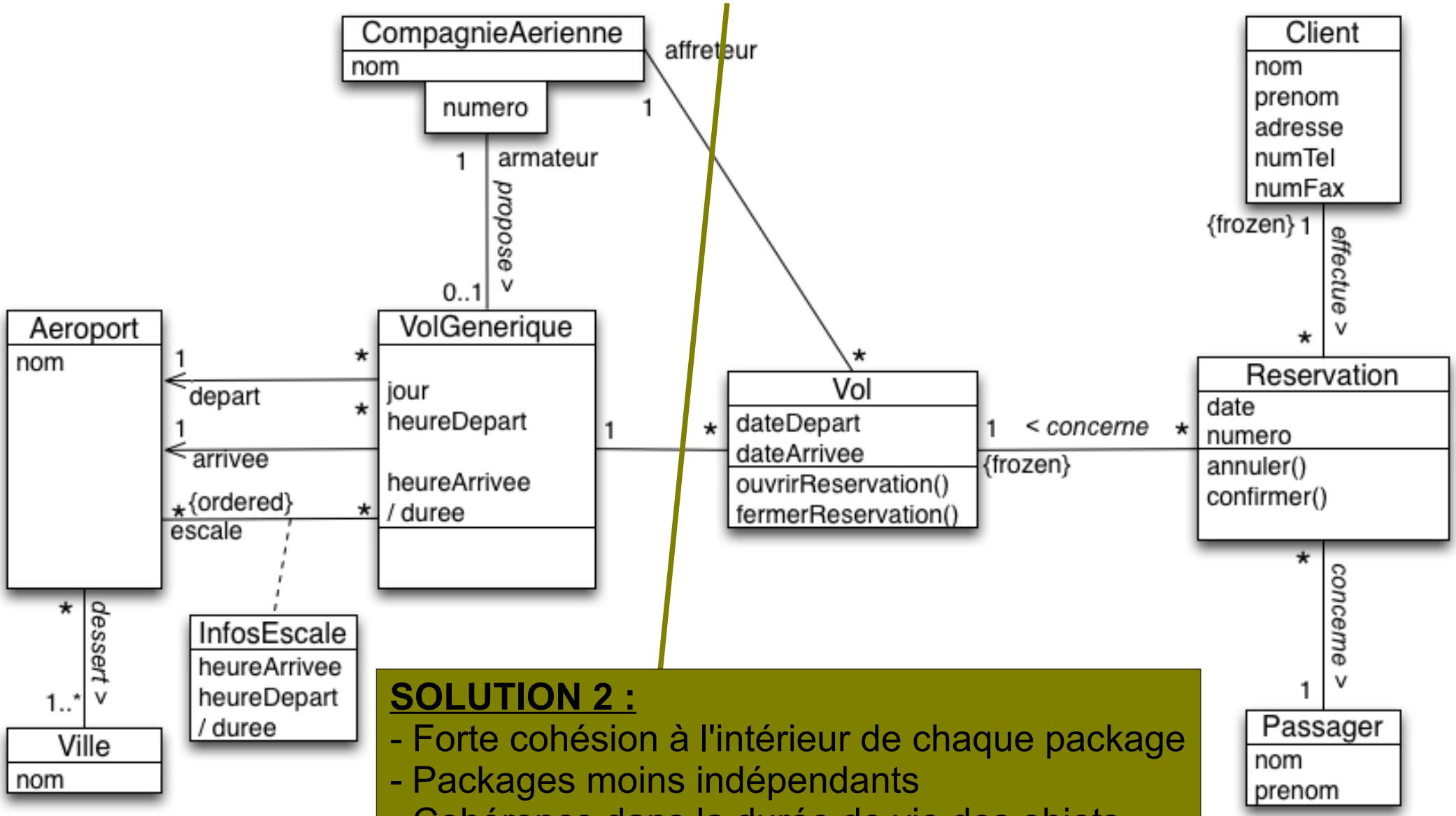
Structuration du modèle statique (packages)



SOLUTION 1 :

- Forte cohésion à l'intérieur de chaque package
- Packages presque indépendants

Structuration du modèle statique (packages)

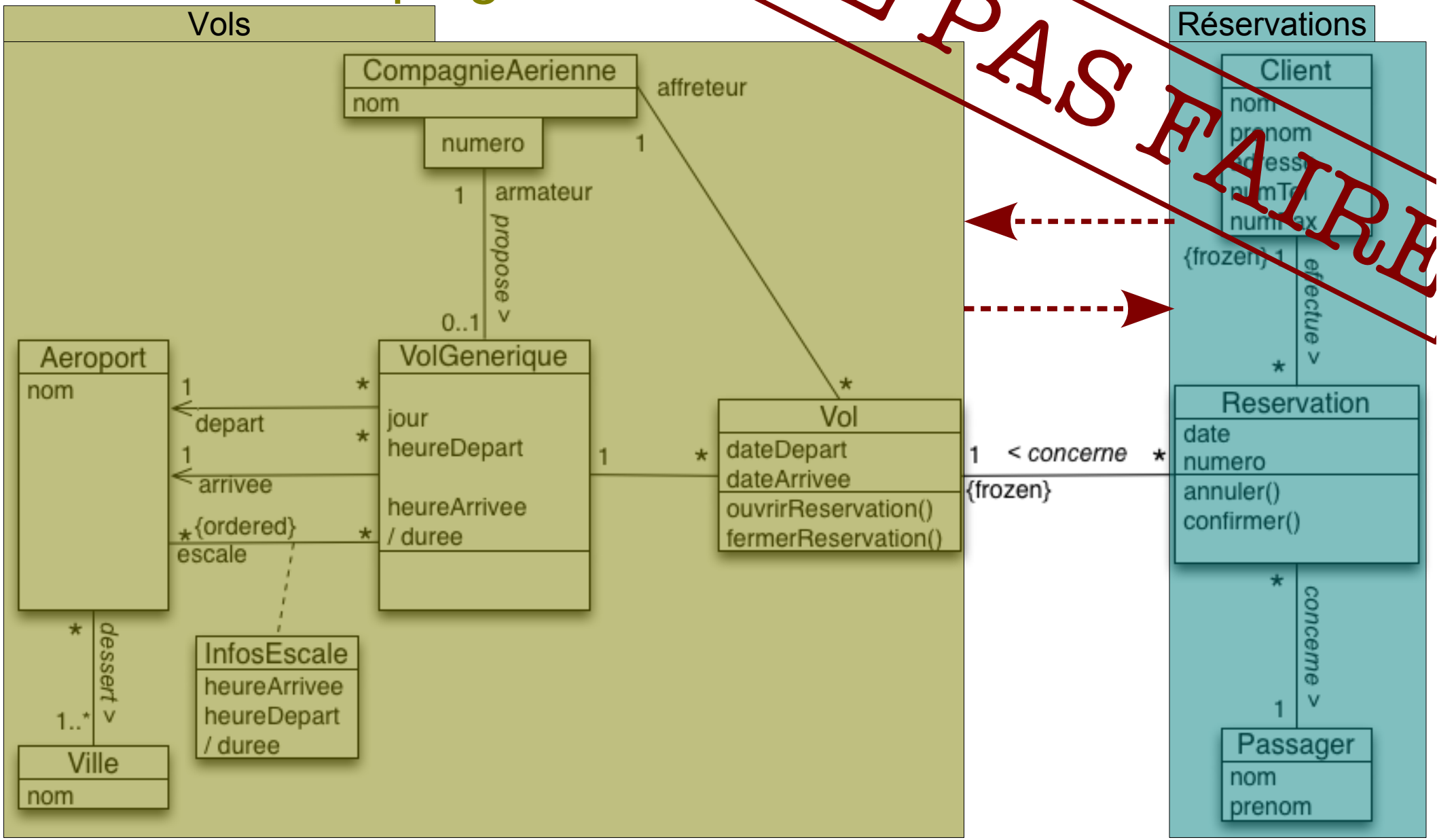


SOLUTION 2 :

- Forte cohésion à l'intérieur de chaque package
- Packages moins indépendants
- Cohérence dans la durée de vie des objets

Etude du couplage : solution 1

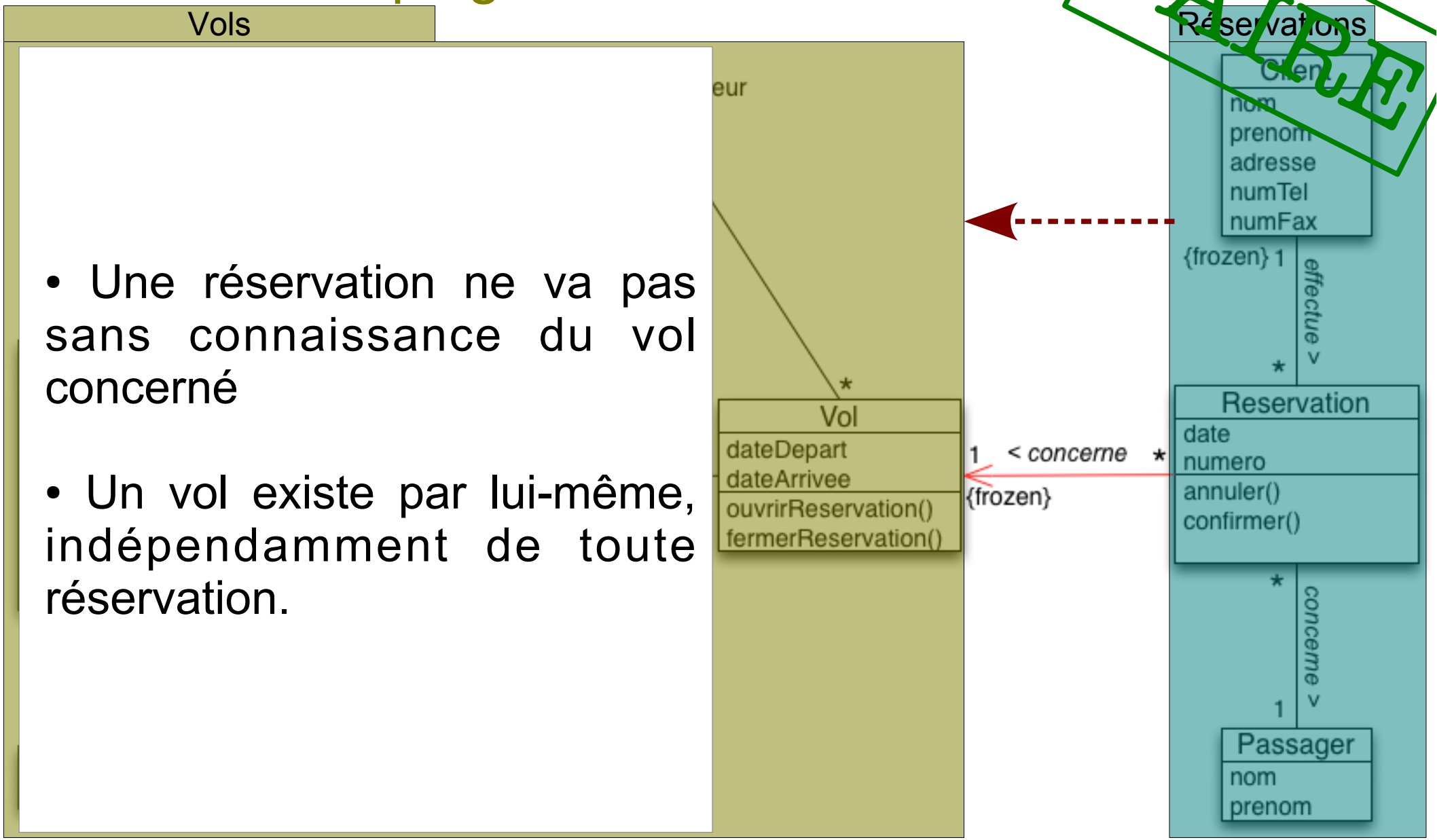
NE PAS FAIRE



Etude du couplage : solution 1

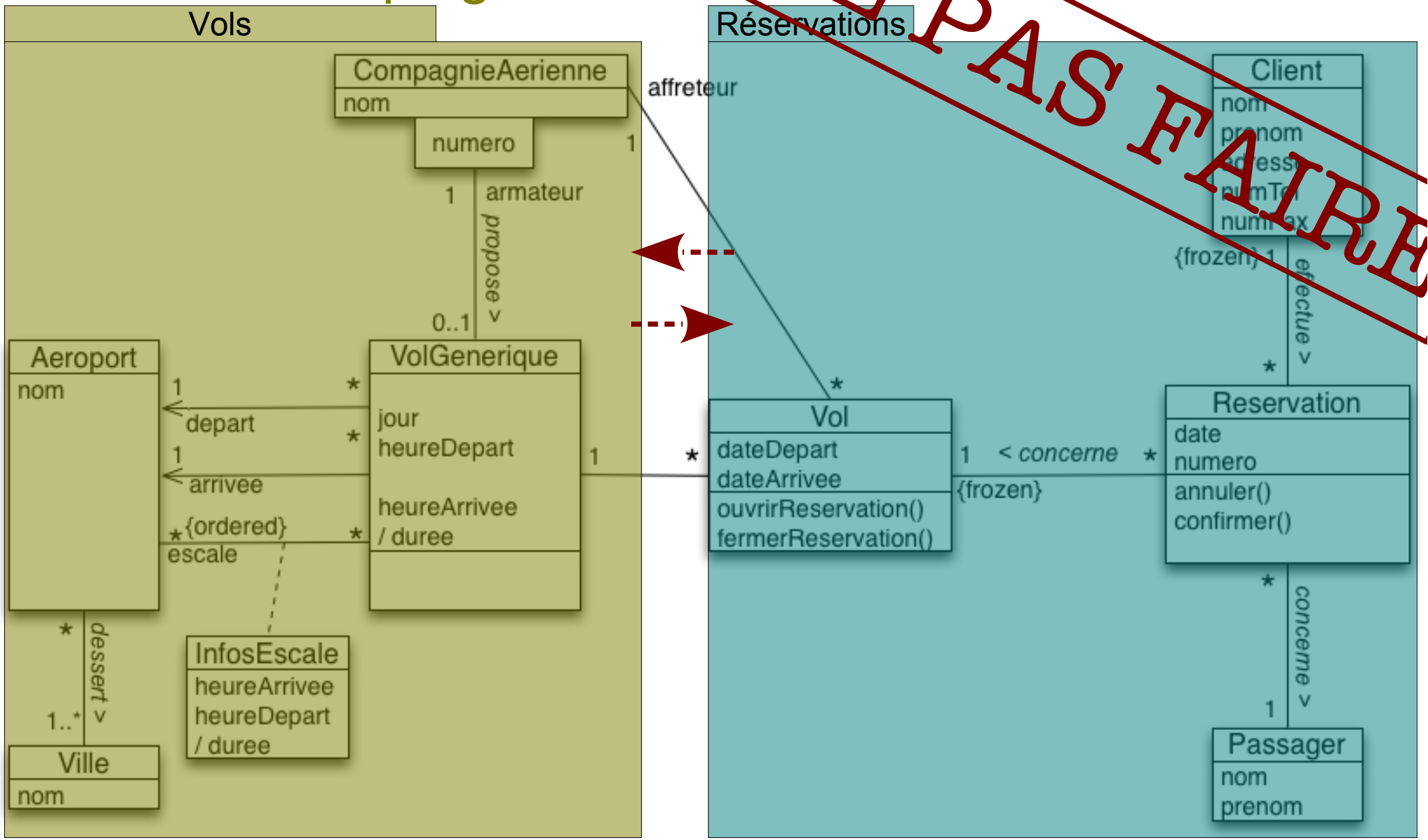
FAIRE

- Une réservation ne va pas sans connaissance du vol concerné
- Un vol existe par lui-même, indépendamment de toute réservation.

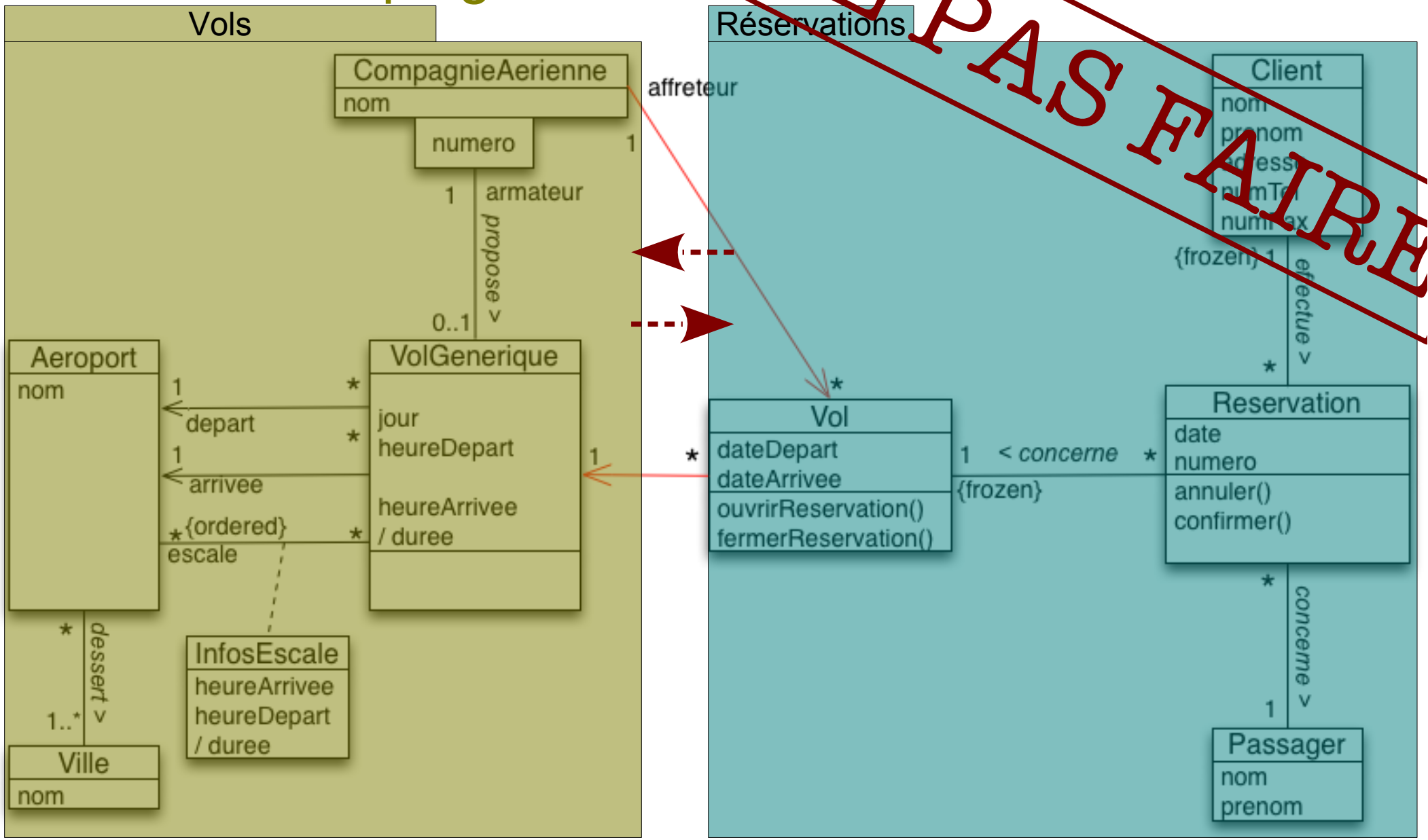


Etude du couplage : solution 2

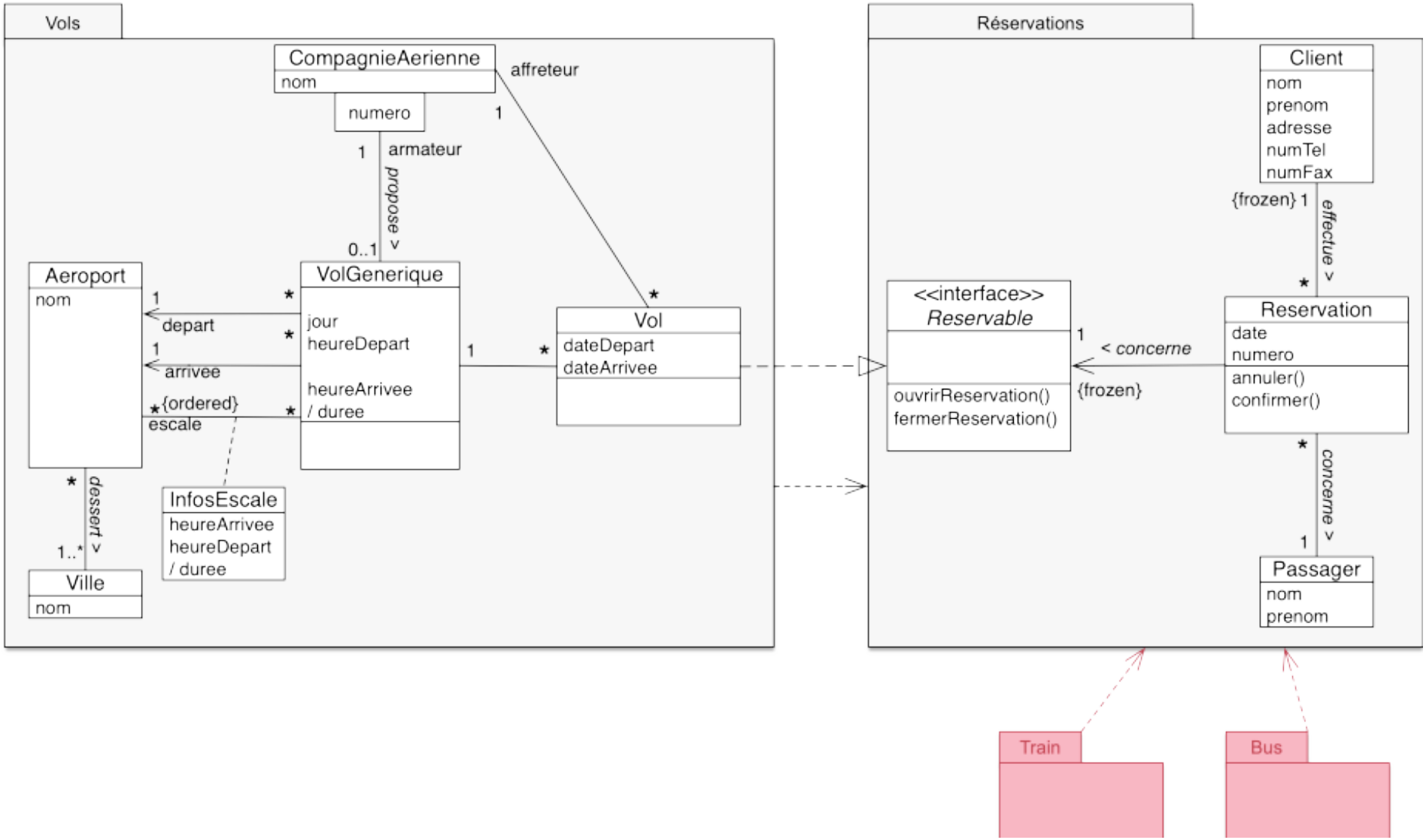
NE PAS FAIRE



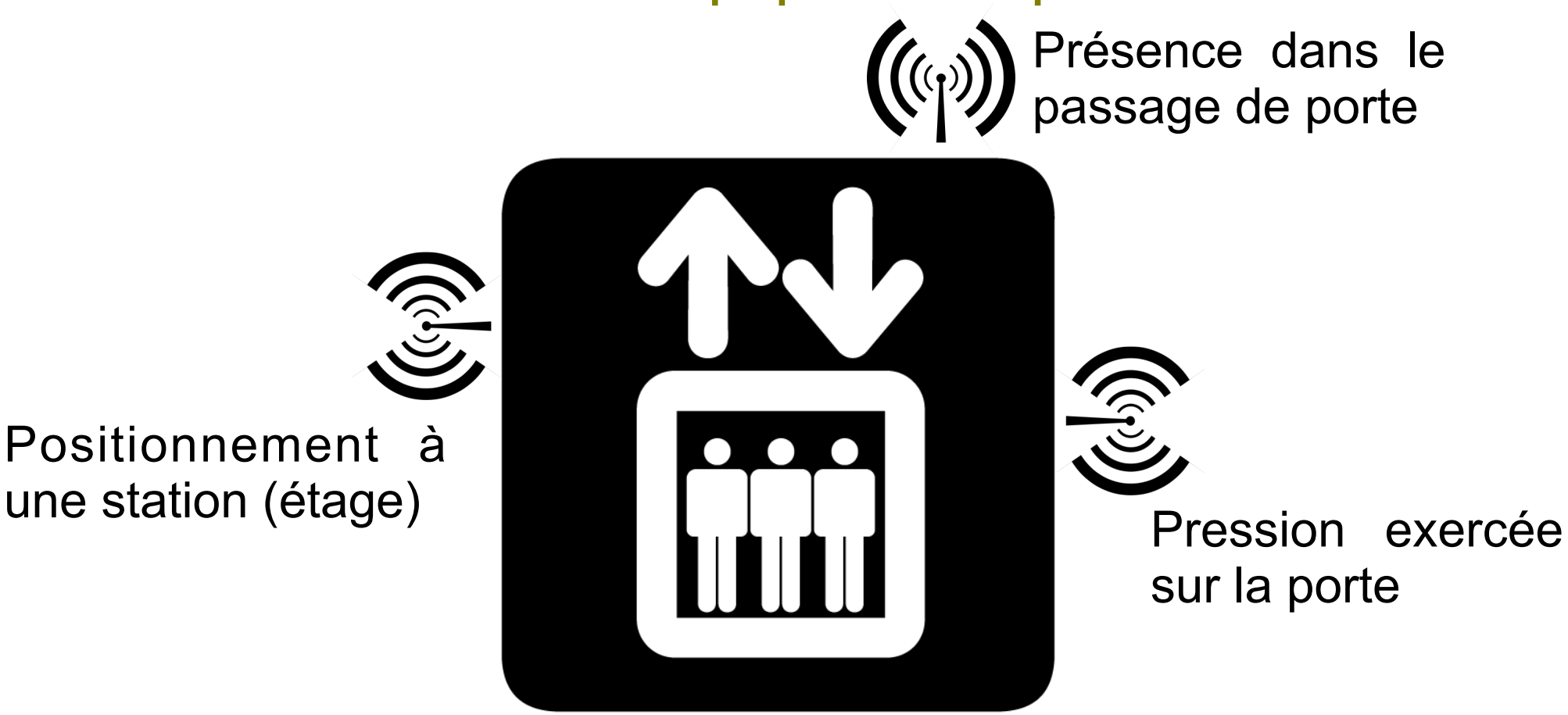
Etude du couplage : solution 2



Solution générique !

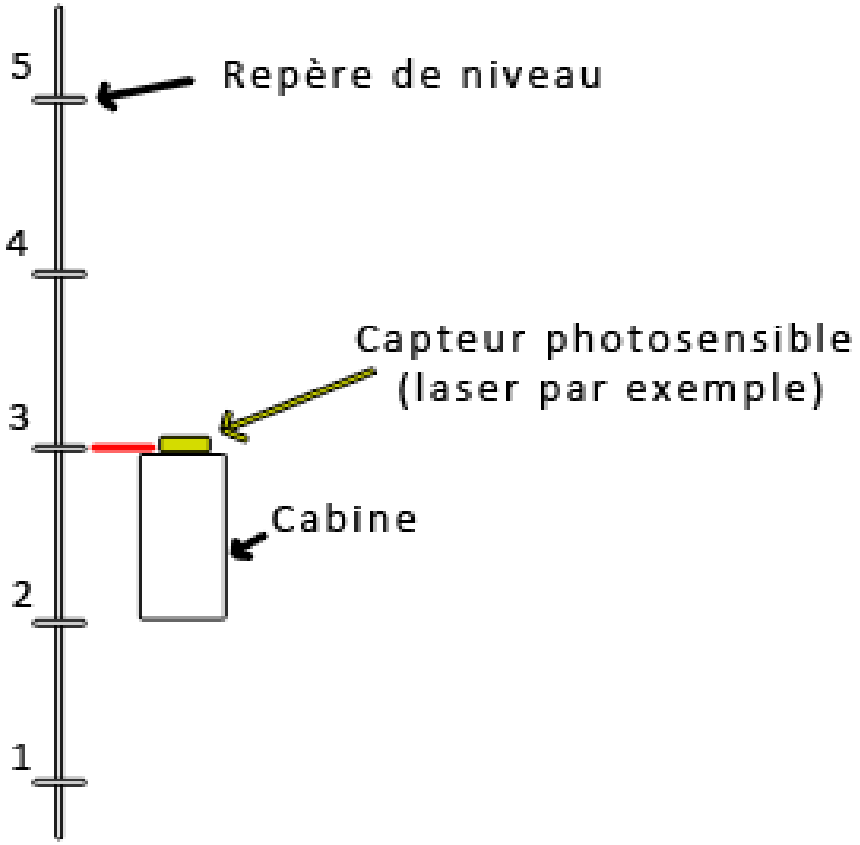


Un ascenseur... c'est équipé de capteurs !



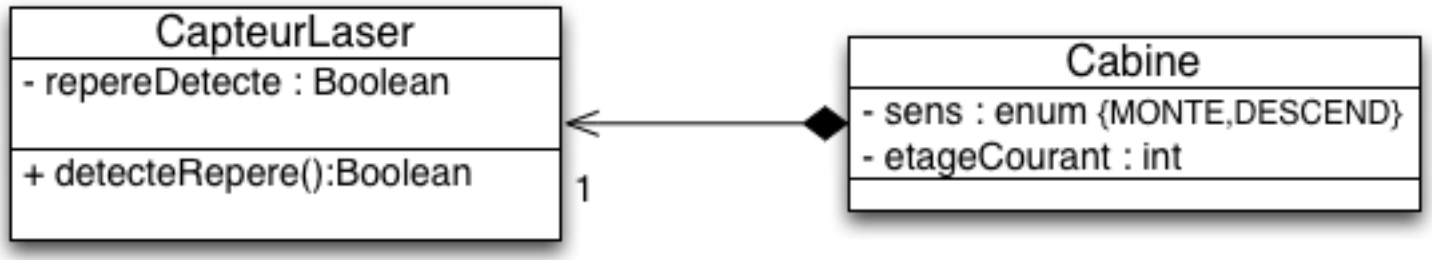
À modéliser

Une cabine possède diverses informations comme son sens courant de déplacement et son étage actuel.
Un capteur photosensible est placé sur la cabine pour indiquer quand celle-ci change d'étage. Il détecte un repère placé à chaque étage.



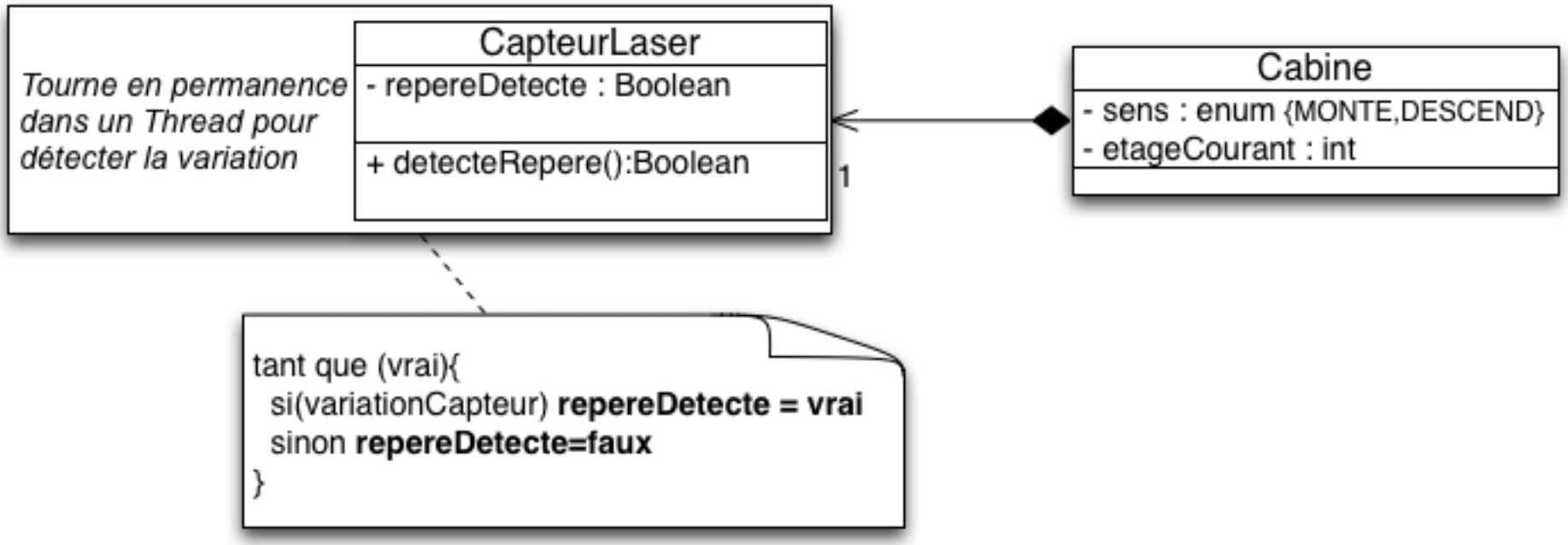
À modéliser

Une cabine possède diverses informations comme son sens courant de déplacement et son étage actuel.
Un capteur photosensible est placé sur la cabine pour indiquer quand celle-ci change d'étage. Il détecte un repère placé à chaque étage.



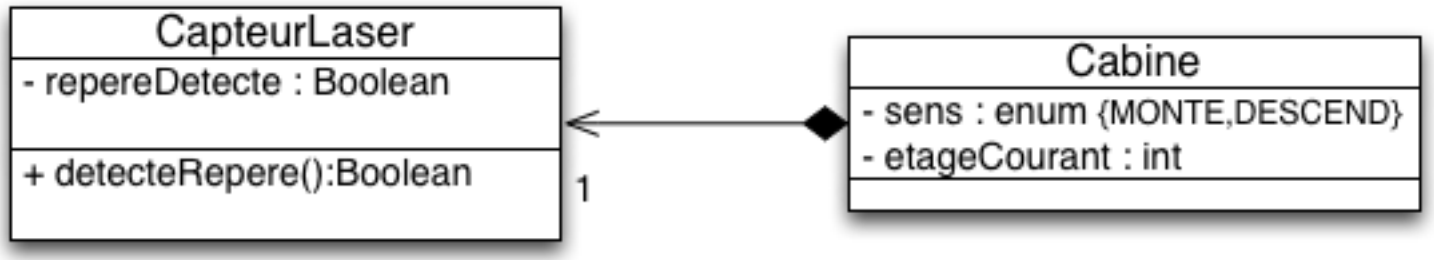
À modéliser

Une cabine possède diverses informations comme son sens courant de déplacement et son étage actuel.
Un capteur photosensible est placé sur la cabine pour indiquer quand celle-ci change d'étage. Il détecte un repère placé à chaque étage.



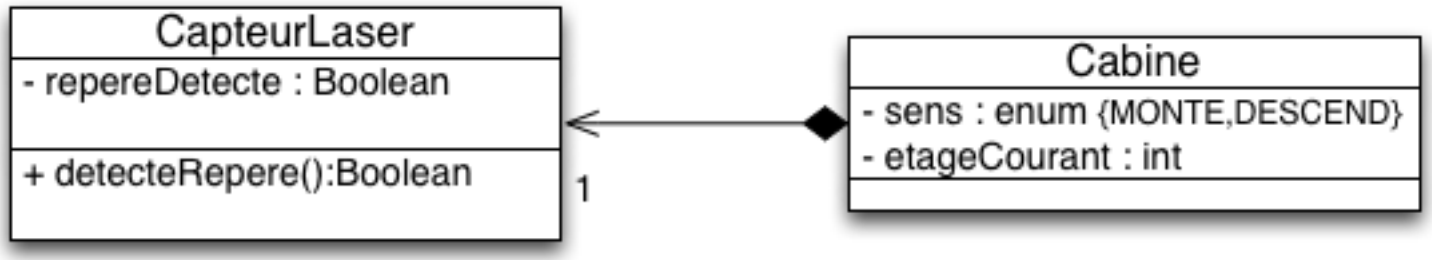
Notons

La cabine peut juste consulter l'état du capteur sans en altérer le comportement. Chaque objet évolue indépendamment.



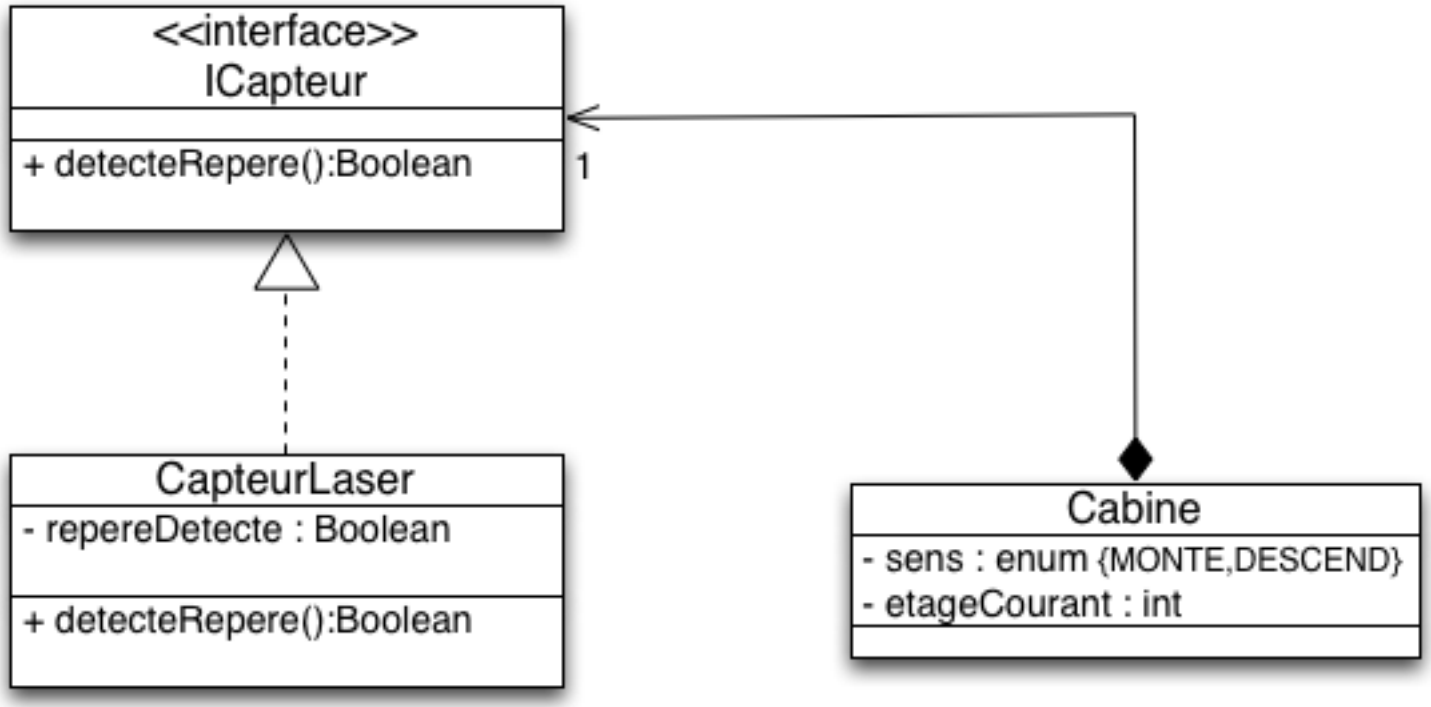
À modéliser

Envisager l'adaptation à n'importe quel type de capteur qui signale juste un changement d'état.



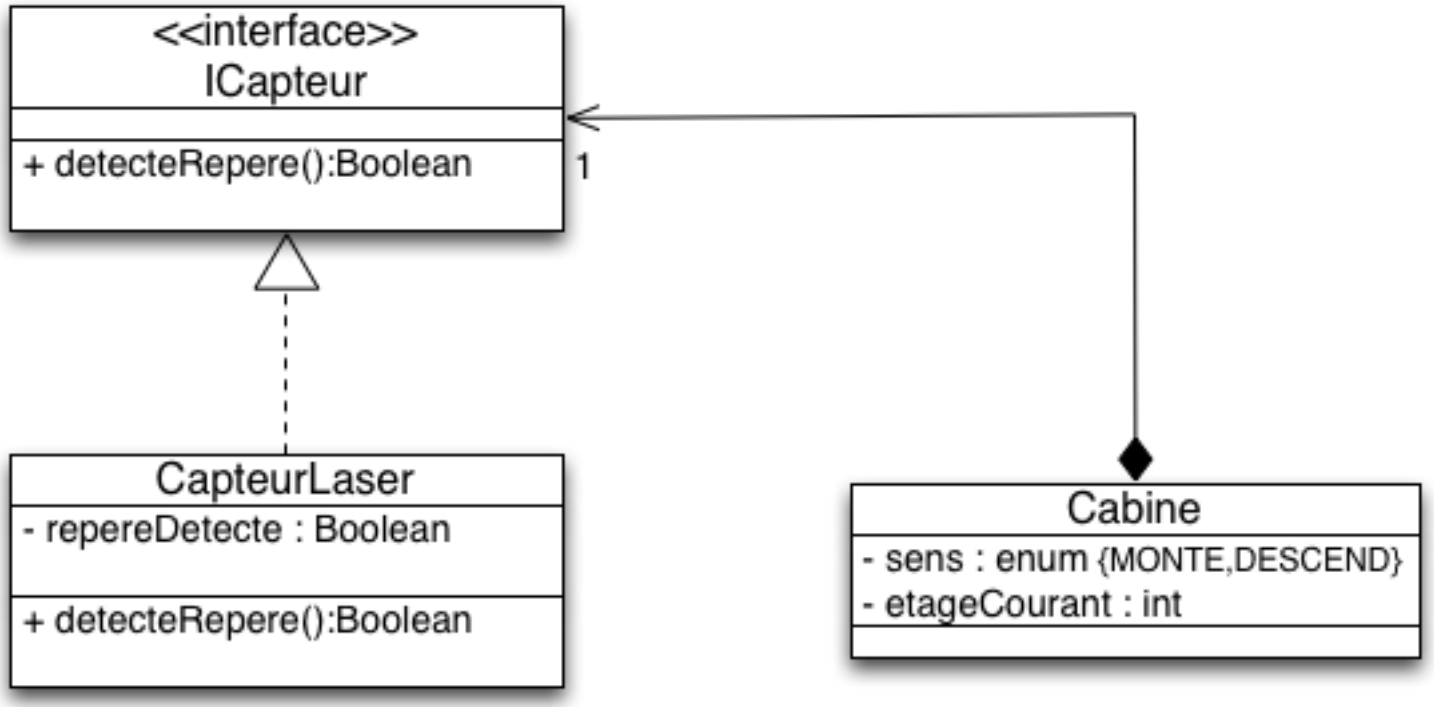
À modéliser

Envisager l'adaptation à n'importe quel type de capteur qui informe juste sur la détection d'un repère.



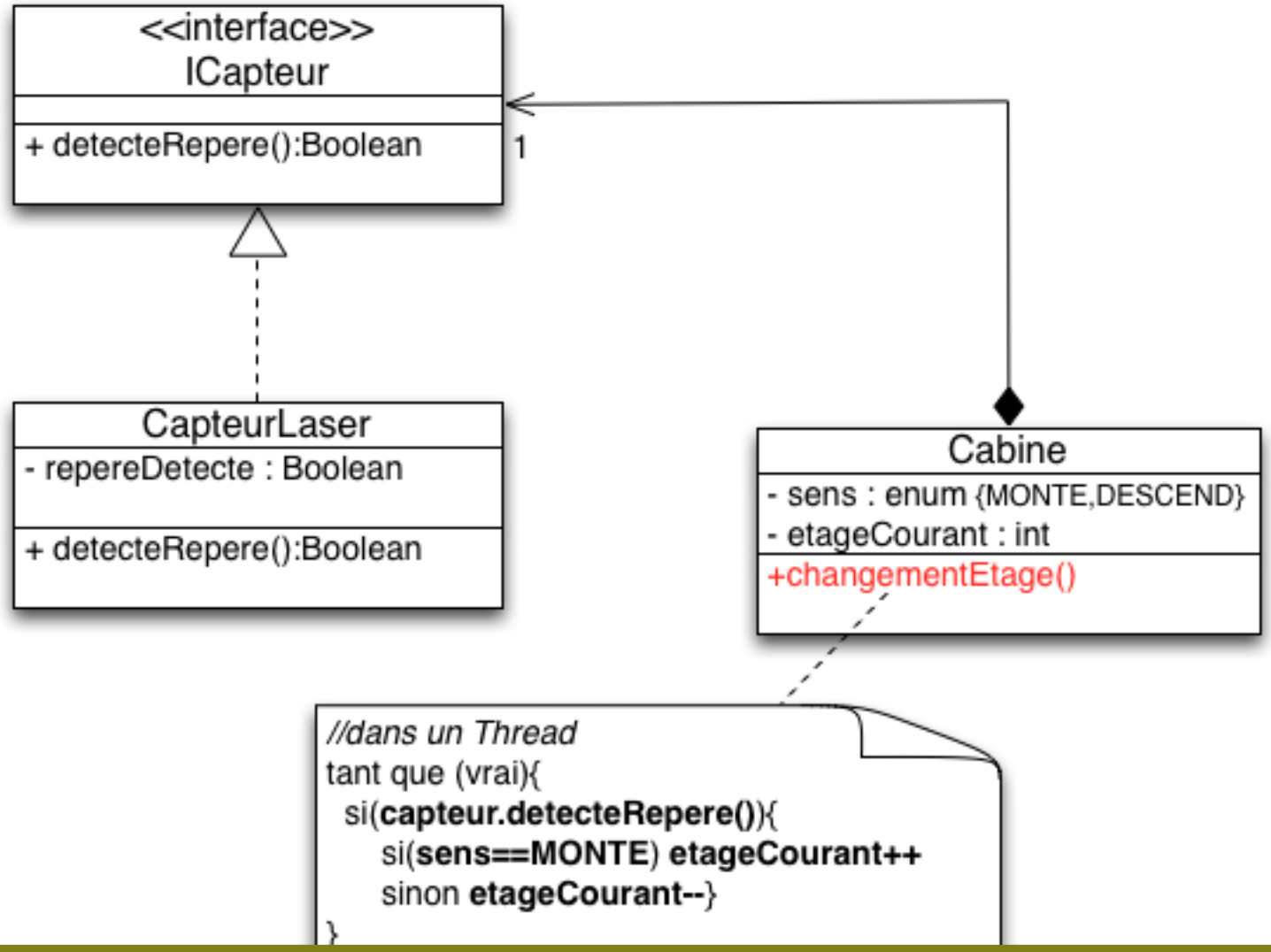
Notons

Modélisation très flexible : si on décide de changer de capteur il suffira de créer une nouvelle classe qui ***réalise*** l'interface.
Le code pour la cabine ne changera pas.



Notons

Limite de la modélisation : *changementEtage()* contiendra une boucle qui interroge en permanence le capteur
→ **complexe et coûteux** !

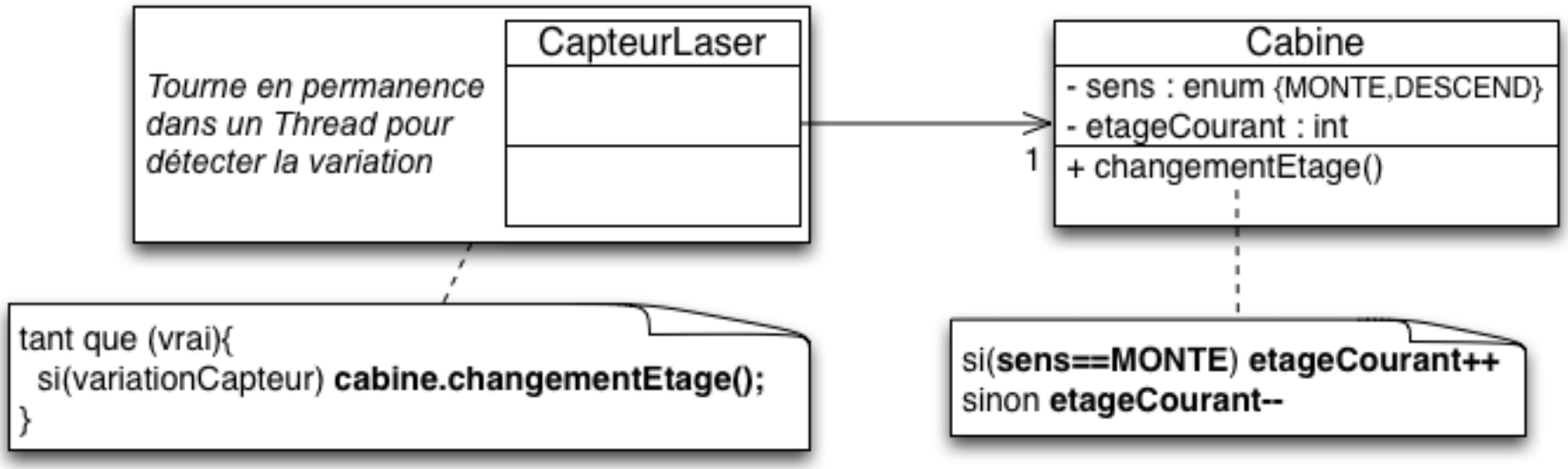


Autre approche :

« le capteur prévient la cabine... »

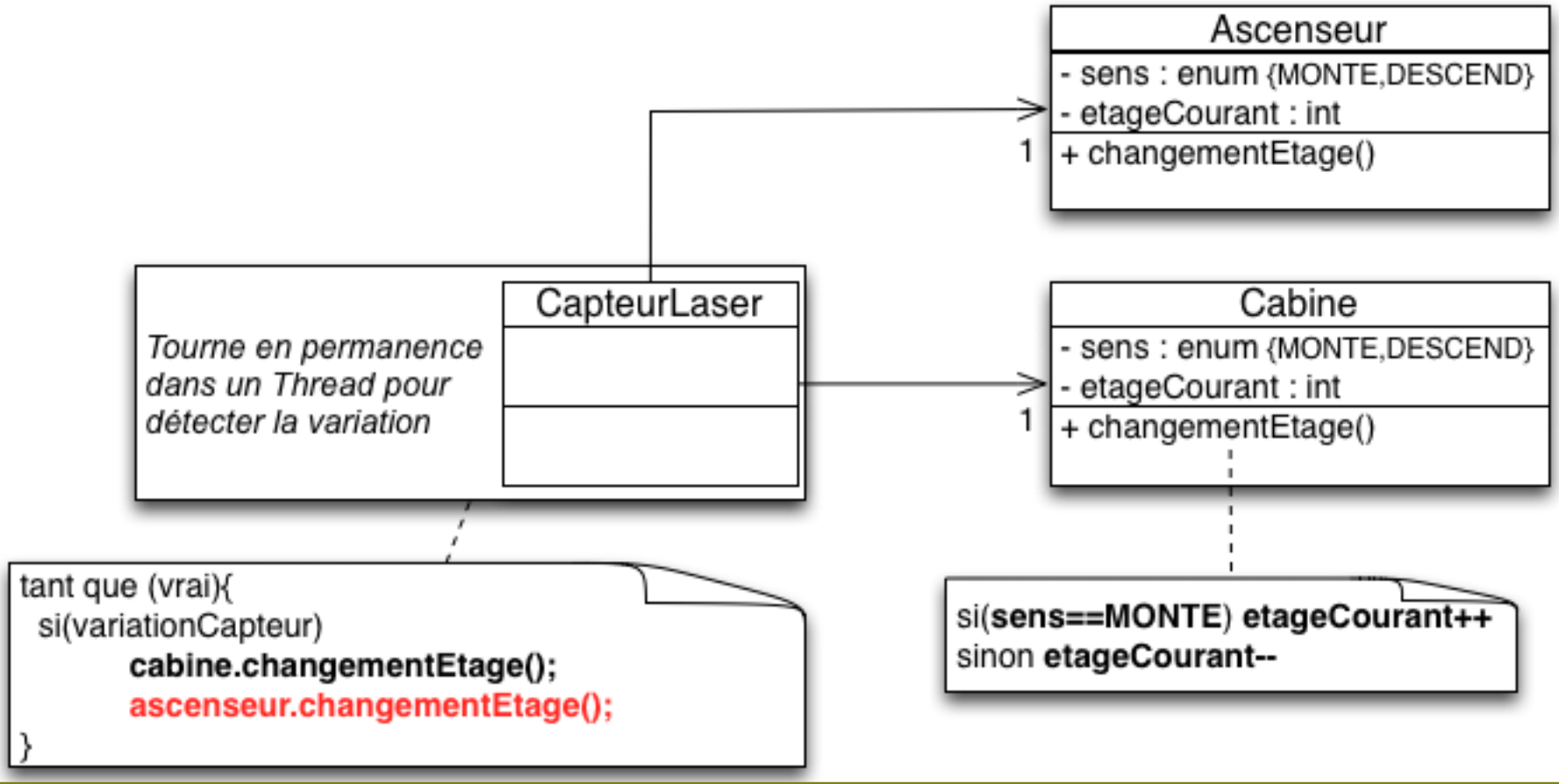
À modéliser

Une cabine possède diverses informations comme son sens courant de déplacement et son étage actuel.
Un capteur photosensible est placé sur la cabine pour indiquer quand celle-ci change d'étage. Il détecte un repère placé à chaque étage.



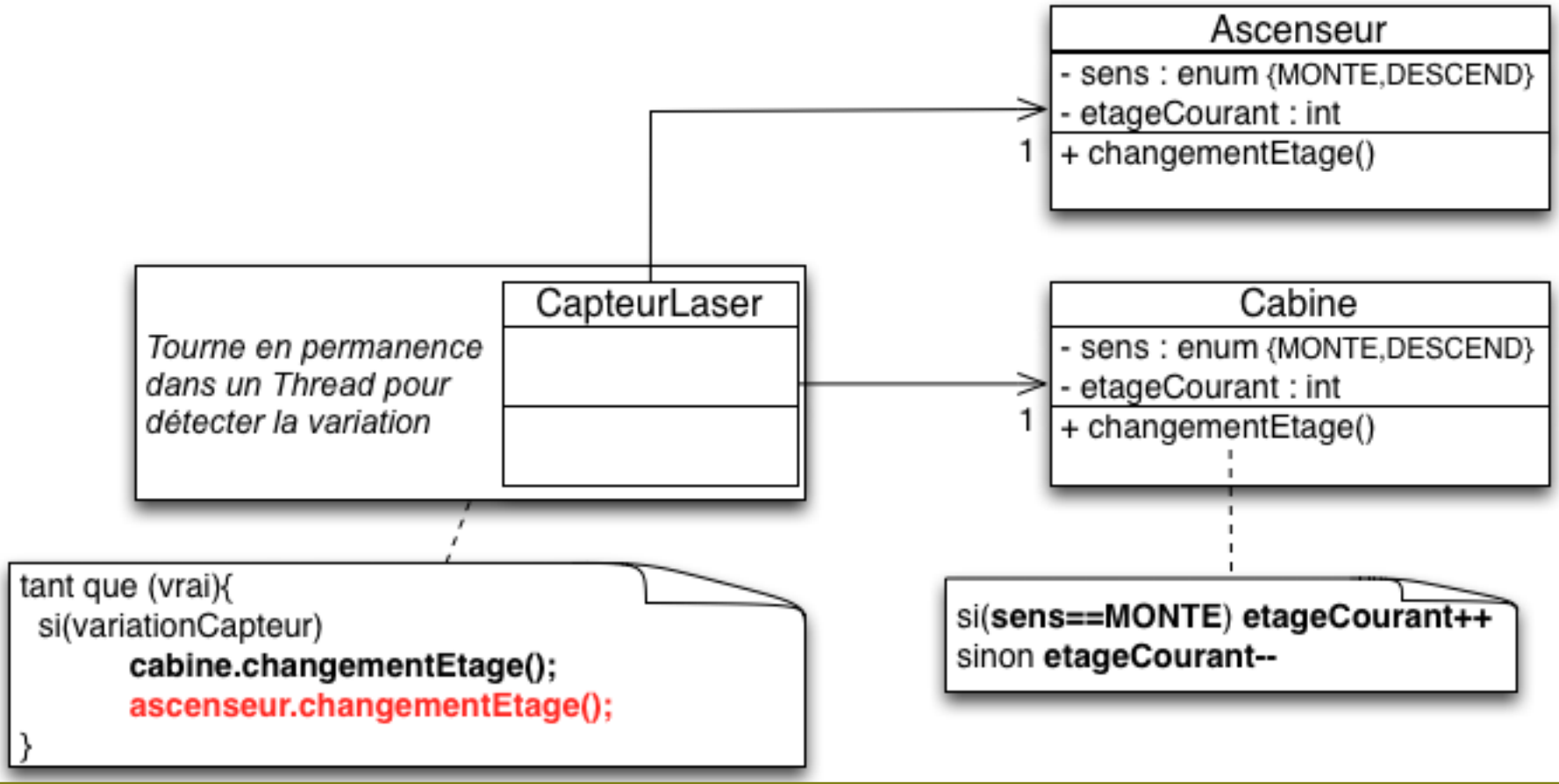
Notons

Plus besoin de boucle dans *changementEtage()*.
→ Et si l'ascenseur a également besoin d'être informé... ?



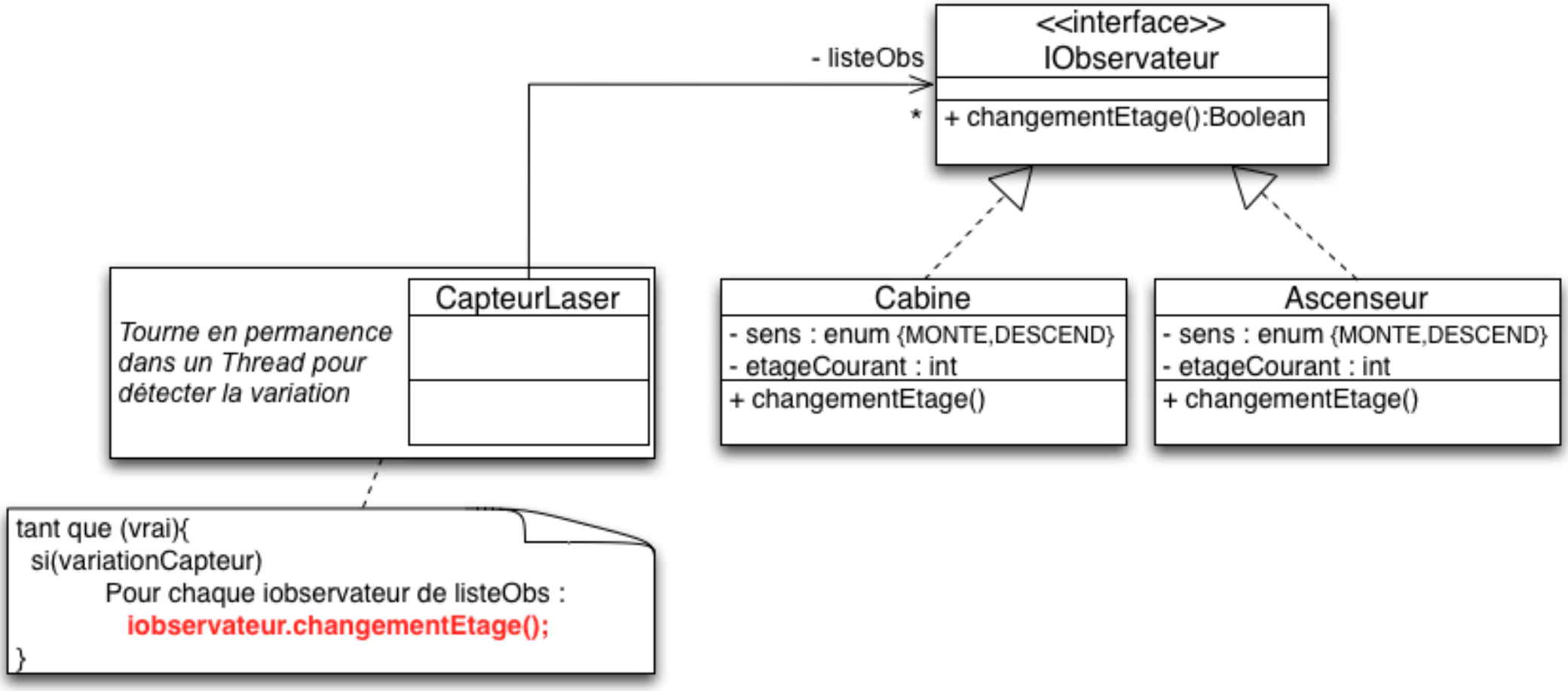
Notons

Plus besoin de boucle dans *changementEtage()*.
→ Et si l'ascenseur a également besoin d'être informé... ?
→ Et si le PC sécurité doit lui aussi être informé... ?



À modéliser

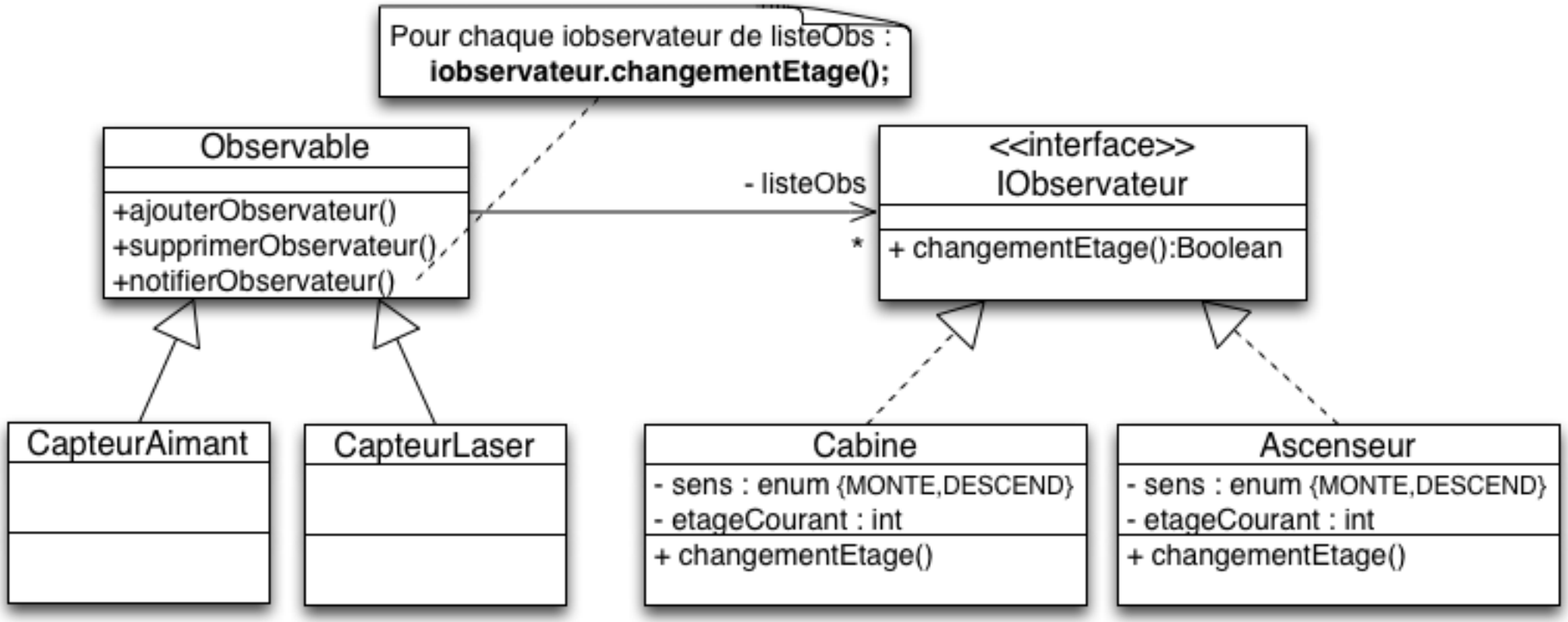
Envisager l'adaptation à n'importe quel type de classe ayant besoin d'être informée



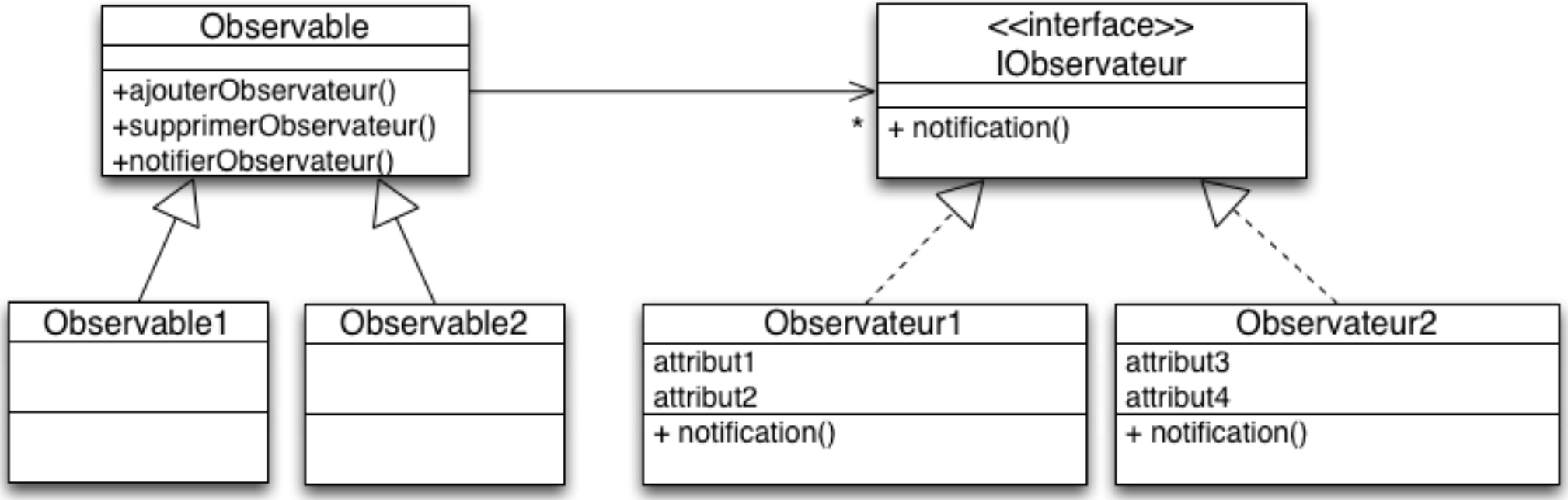
Un dernier effort...

À modéliser

Envisager l'adaptation à n'importe quel type de capteur

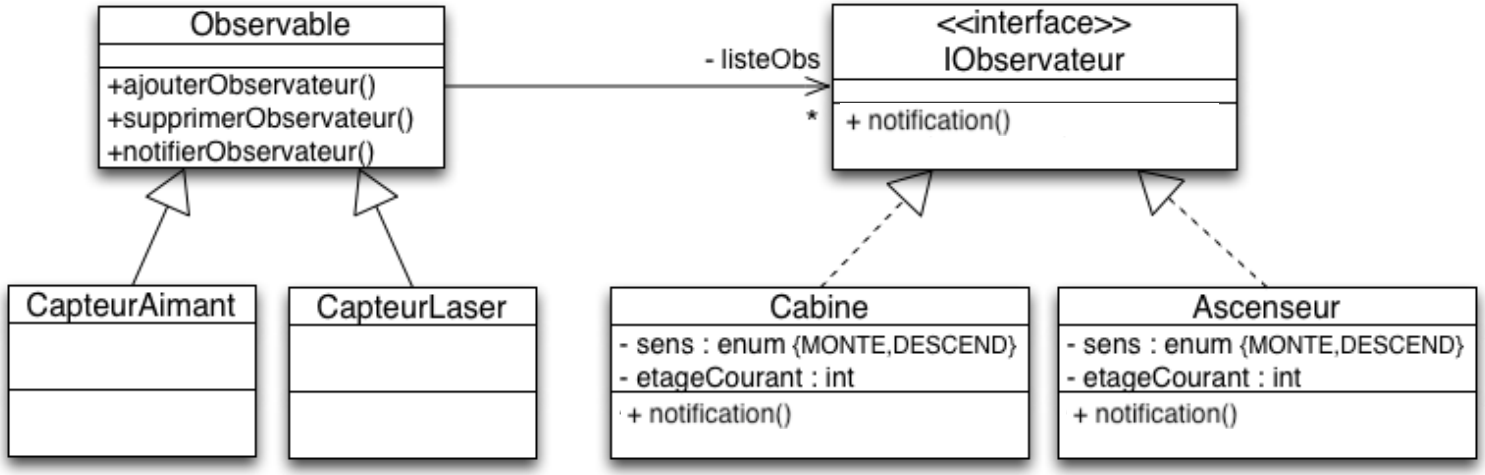


Pattern « observateur »

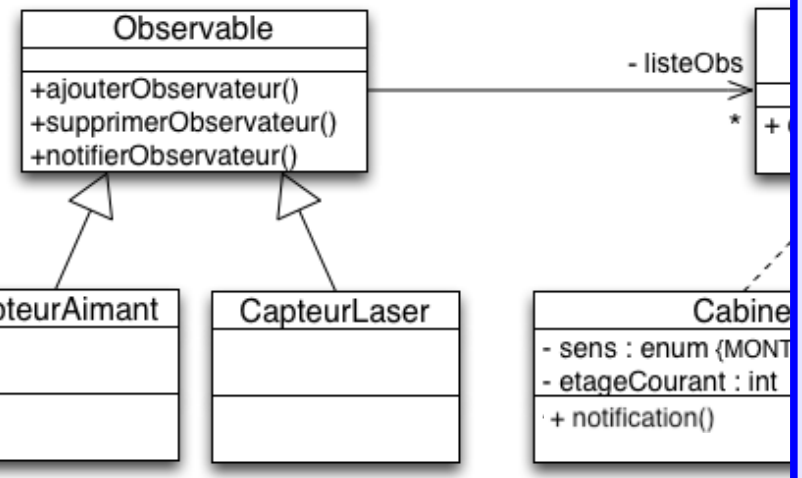


Pattern « observateur » exemple

```
interface IObservateur{
      public void notification() ;
}
```



Pattern « observateur »




```
class Observable{

 private List<IObservateur> listeObs ;

 public Observable(){
   listeObs=new LinkedList<IObservateur>();
 }

 public void notifierObservateurs(){
   Iterator<IObservateur> it=listeObs.iterator() ;
   while(it.hasNext()){
     IObservateur obs=it.next ;
     obs.changementEtage() ;
     }

 public void ajouterObservateur(IObservateur o){
   listeObs.add(o) ;
 }

 public void supprimerObservateur(IObservateur o){
   listeObs.remove(o) ;
 }
}
```

Questions ?