École de technologie supérieure
Département de Génie logiciel et des TI

Document thématique

OCL en 7 exemples – Partie III: le forage des classes et l'expression des invariants

---

Auteur(s) : Sylvie Ratté
Date de création : 9 février, 2005
Réviseurs : Sylvie Ratté
Date de révision : 15 mars, 2005

Références

A special thanks to Math Dwyer from University of Nebraska, John Hatcliff and Rod Howell from Kansas State University (KSU) to have given me the authorization to use and adapt their class material. The examples presented are originally theirs. The chronological order in which the subjects are presented is inspired by their slides for the course CIS771: Software Specification at KSU.

1 Introduction

Dans les versions précédentes, il n'était pas possible d'exprimer le fait qu'un cycle de pré-requis ne peut exister. On cherche en fait à établir une fermeture transitive pour parcourir les cycles. EN d'autres termes, on cherche à vérifier si x est un prérequis de y et que y est un prérequis de z, x est un prérequis de z.

2 Fermeture transitive

En OCL, il n'existe pas de fermeture transitive comme en Z (en Z : si x R y et y R z, x R* z). Par contre, OCL admet les définitions récursives.

2.1 Première tentative

Fermeture-1.use

```
class A
operations
  fermeture() : Set(A) = succ.fermeture()->asSet()->including(self)
end
association R between
  A[*] role pred
```

```
  A[*] role succ
end
```

Instantiation possible (fermeture-instantiation-1.cmd):

```
!create a1:A
!create a2:A
!create a3:A
!insert (a1,a2) into R
!insert (a2,a3) into R
use> ? a1.fermeture()
-> Set{@a1,@a2,@a3} : Set(A)
```

Qu'est-ce qui se passé exactement lorsque a1.fermeture() est évalué? Examinons la trace.

```
Level 1 call: self = a1, a1.succ = a2
Level 2 call: self = a2, a2.succ = a3
Level 3 call: self = a3, a3.succ = {}
Level 3 return: Set{@a3}
Level 2 return: Set{@a2,@a3}
Level 1 return: Set{@a1,@a2,@a3}
```

Qu'arrivera-t-il? (fermeture-instantiation-2.cmd)

```
!create a1:A
!create a2:A
!create a3:A
!insert (a1,a2) into R
!insert (a2,a3) into R
!insert (a3,a1) into R
use> ? a1.closure()
…java.lang.RuntimeException: StackOverflow…
```

2.2 Deuxième tentative

Le problème évidemment vient du fait que la récursivité n'est pas stoppée. Intuitivement, on doit s'arrêter lorsque l'on a ramassé tous les éléments rencontrés et que l'on revient à la valeur initiale (ici « a1 »). En d'autres mots, on doit s'arrêter lorsqu'on ne rencontre plus rien de « neuf » en parcourant la relation.

Heureusement, OCL admet l'utilisation de « If-then-else » dont la syntaxe est :

```
if expr-bool then expr1 else expr2 endif
```

- Retourne expr1 si « bool-expr » est vraie.
  Retourne expr2 si « bool-expr » est fausse.
- Est indéfinie si « bool-expr » n'est pas définie.

La bonne définition de « fermeture » serait:

```
fermeture(s : Set(A)) : Set(A) =
if s->includesAll(s.succ->asSet) then s
else fermeture(s->union(s.succ->asSet))
endif
Note: La fermeture est réflexive car l'argument "s" doit être
inclus dans le résultat.
```

L'appel initial serait:

```
atteignableDeSelf() : Set(A) = fermeture(Set{self})
```

Fermeture-2.use

```
class A
operations
  fermeture(s : Set(A)) : Set(A) =
    if s->includesAll(s.succ->asSet) then s
    else fermeture(s->union(s.succ->asSet))
    endif
  atteignableDeSelf() : Set(A) = fermeture(Set{self})
end
association R between
  A[*] role pred
  A[*] role succ
End
```

```
Essayons maintenant avec: a1.atteignableDeSelf() ?
  Level 1 call: s = {@a1}, s.succ = {@a2}
  Level 2 call: s = {@a1,@a2}, s.succ = {@a2,@a3}
  Level 3 call: s = {@a1,@a2,@a3}, s.succ = {@a1,@a2,@a3}
  Level 3 return: Set{@a1,@a2,@a3}
  Level 2 return: Set{@a1,@a2,@a3}
  Level 1 return: Set{@a1,@a2,@a3}
```

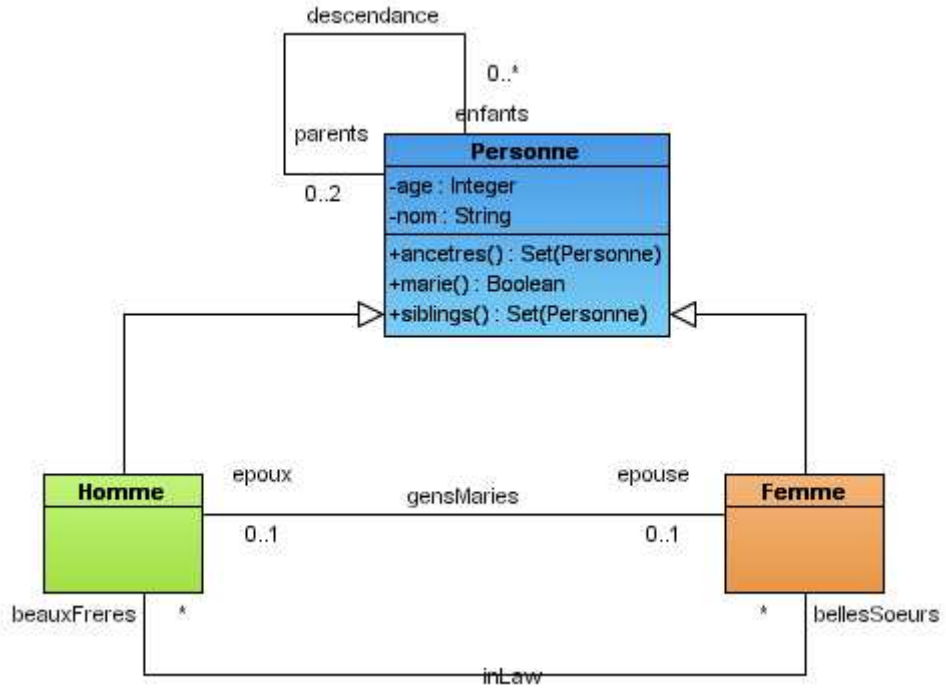
3 Forage des classes

Identification des contraintes

OCL fournit un langage pour définir les invariants mais d'où viennent-ils? Tout dans le diagramme de classes doit être examiné :

- Les relations entre sous-classes;
- Les attributs;
- Les opérations
- Les associations comportant des multiplicités
- Les associations et les attributs dérivés
- Les relations qui existent entre certaines associations/attributs

3.1 UML avancé



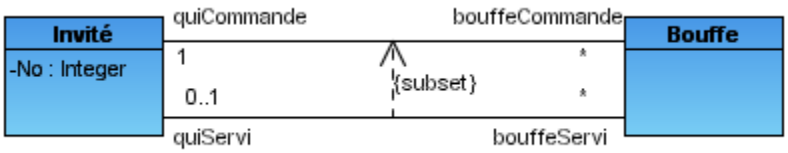
Les méthodes dérivées:

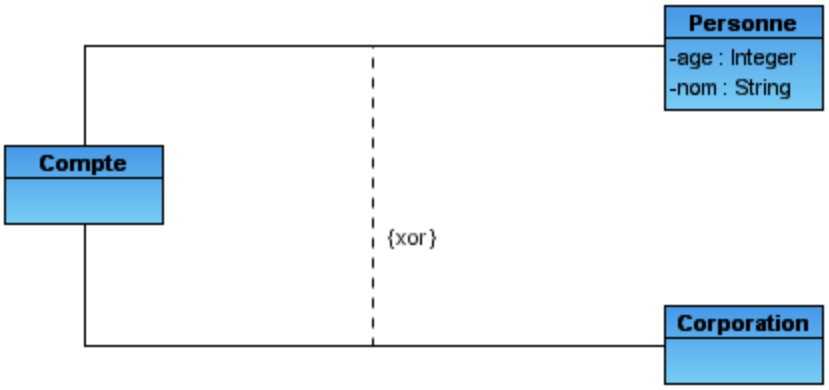
- Ancetres
- Siblings

Les associations dérivées:

- « inLaw »
  Exemple en OCL :

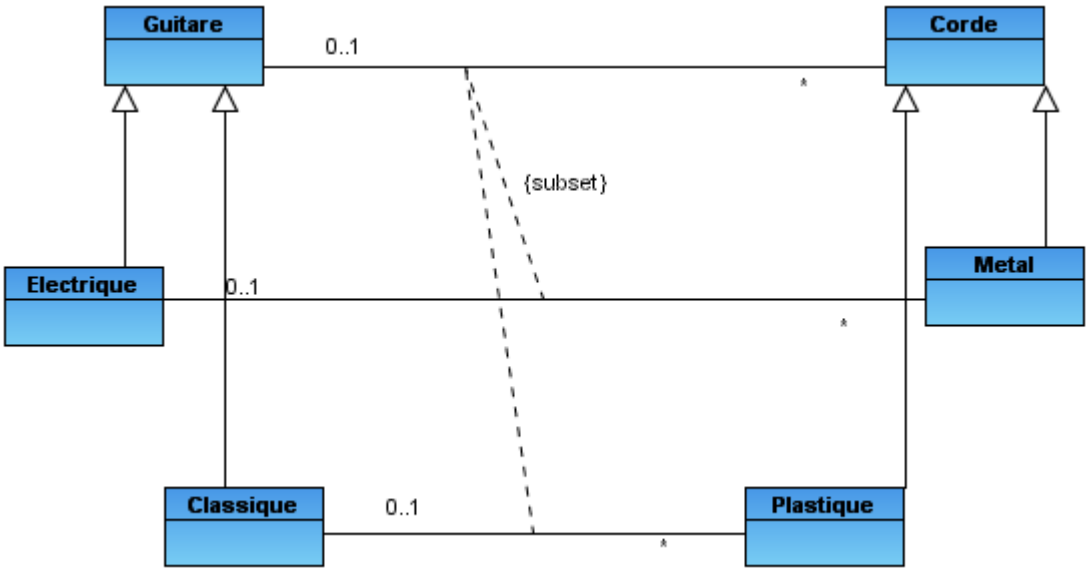
```
context w:Femme

inv beauFrere:
w.beauFrere = w.epoux.parents.enfants->
   select(p | p.oclIsKindOf(Homme) and p <> w.epoux)->
      union(w.parents.enfants->
         select(s | s.oclIsKindOf(Femme) and s <> w)->
            collect(s | s.oclAsType(Femme).epoux))->asSet
```



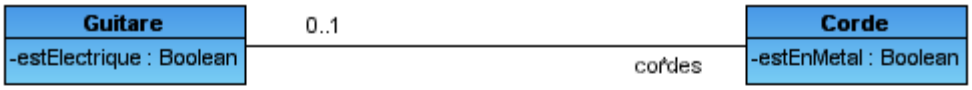


Tous ces aspects peuvent également être capturés en OCL.

La decision devient la vôtre: faire un modèle détaillé…



… ou introduire plus de contraintes.




```
context Guitar
inv ElectriqueMetal:
estElectrique implies cordes->forAll(s | s.estEnMetal)
inv ClassiquePlastique:
not estElectrique implies cordes->forAll(s | not s.estEnMetal)
```

3.2 Contraintes non présentes dans les diagrammes

Certaines contraintes (classes abstraites, sous-classes, multiplicités) peuvent être spécifiées complètement dans le diagramme de classes alors que d'autres peuvent être spécifiées partiellement ou pas du tout dans le diagramme.

Contrainte d'unicité

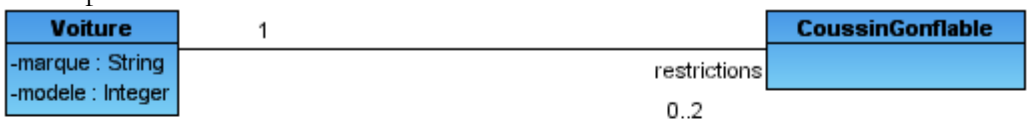
Exemple: les étudiants et les professeurs ont des numéros d'identification unique.

```
context Personne
 inv UniqueIds:
   Personne.allInstances->
      forAll(p1,p2 | p1.id <> p2.id implies p1 <> p2)
```

La question à poser est la suivante: « est-ce que l'égalité implique l'identité? »

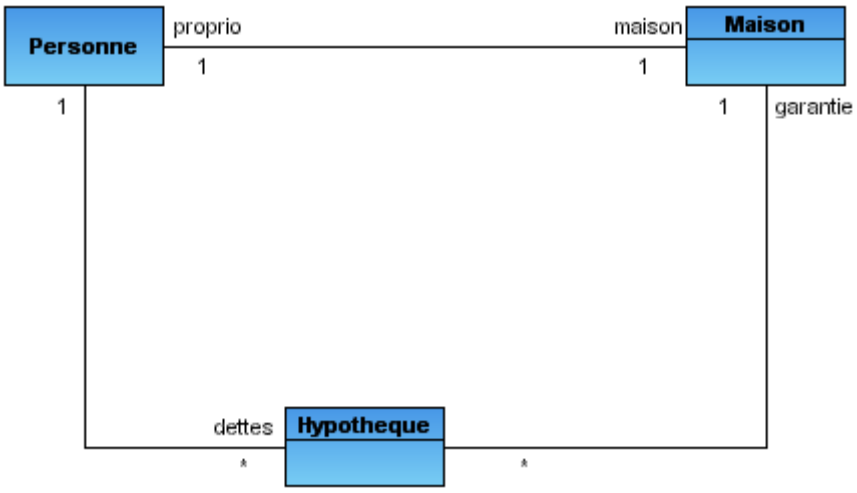
Multiplicités optionnelles

Parfois la valeur de certains attributs ou de certaines associations détermine la taille d'une association partuculière.



Cycles dans les diagrammes de classes

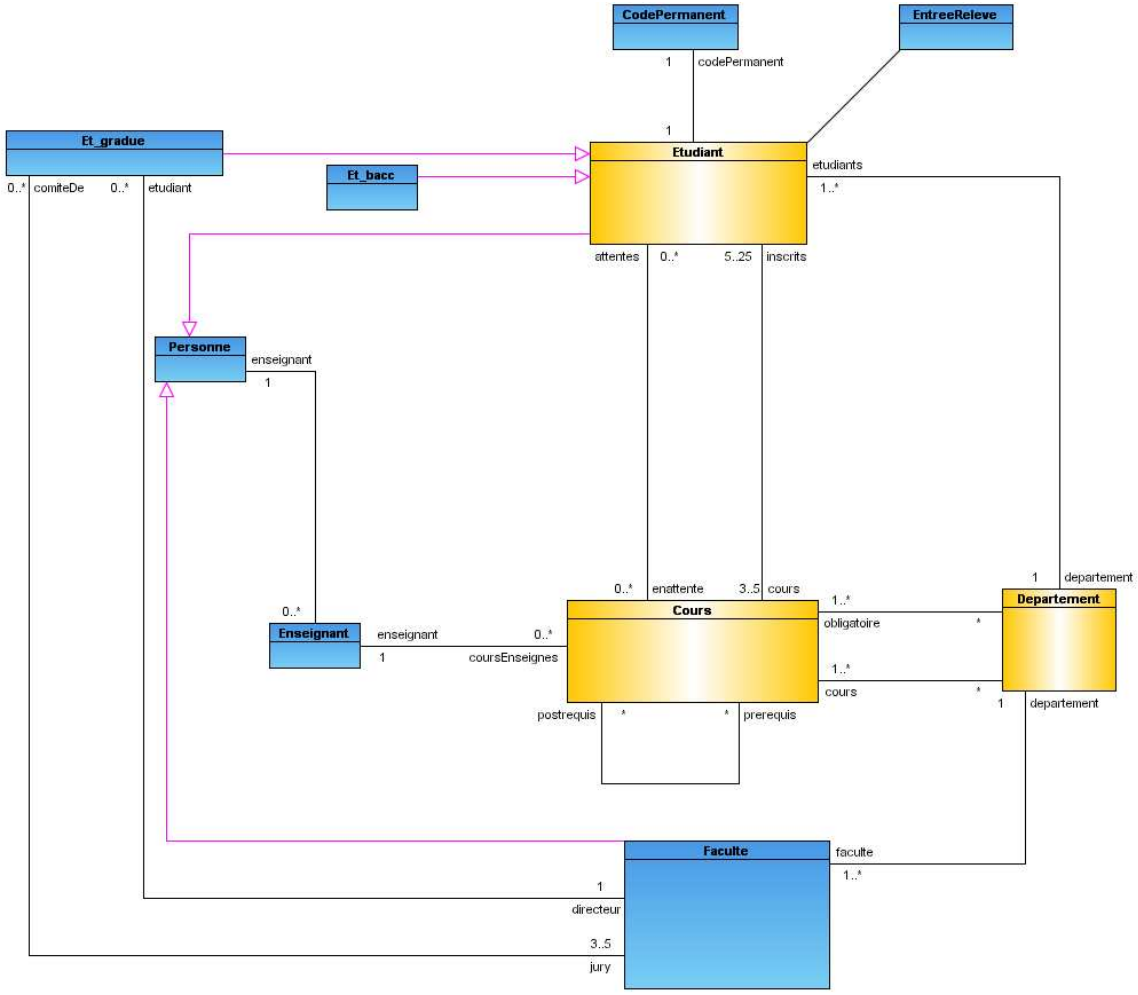
La présence de cycles indique qu'un objet est relié à lui-même. Cela peut être acceptable, inacceptable ou nécessaire. Tous les cycles doivent être examinés.



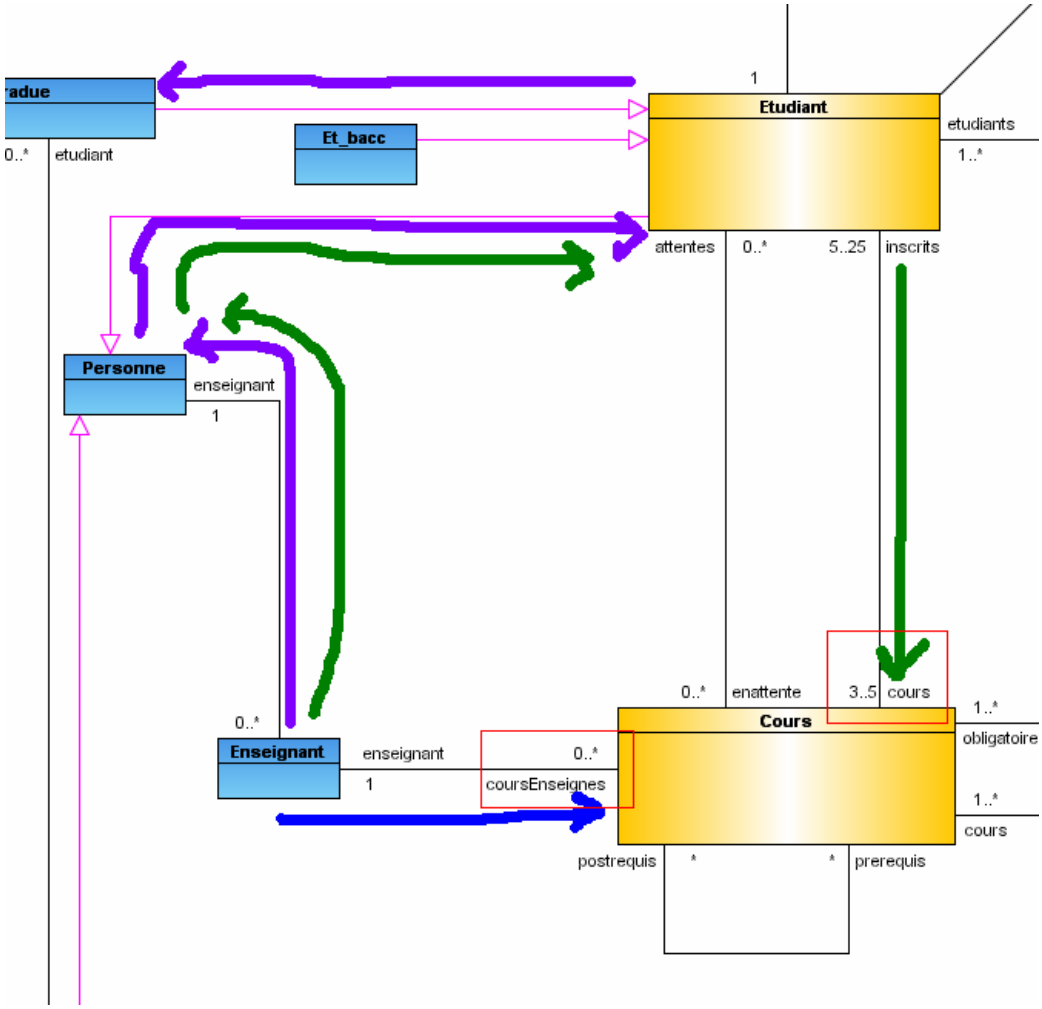

```
context Personne
   inv: dettes.garantie.proprio->forAll(p | p = self)
```

Un exemple plus complet

Pour toutes les contraintes qui suivent, nous partirons du diagramme de classes suivant:

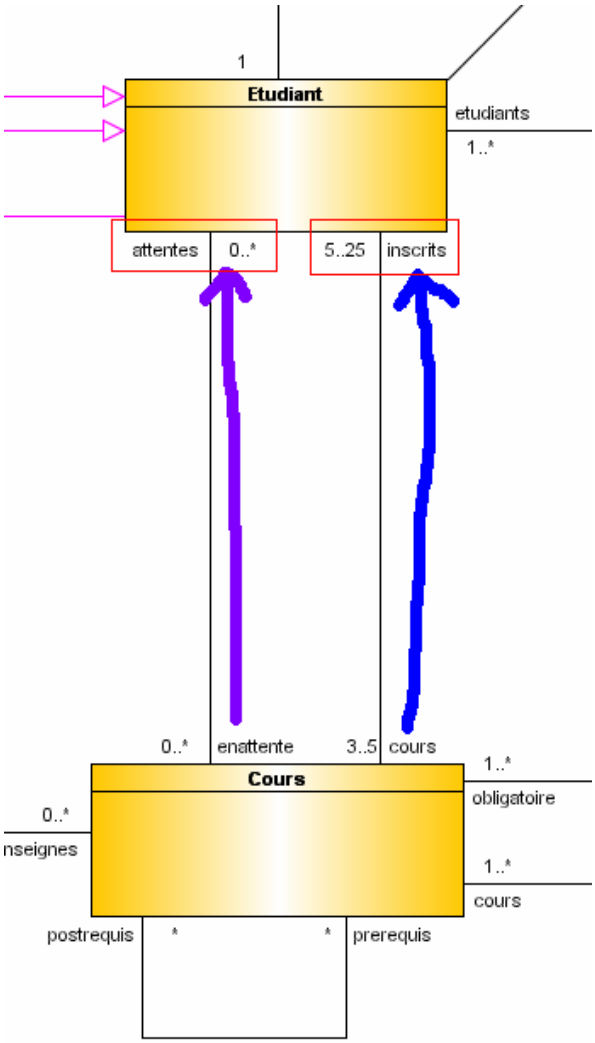


Un étudiant ne peut enseigner un cours auquel il est inscrit.



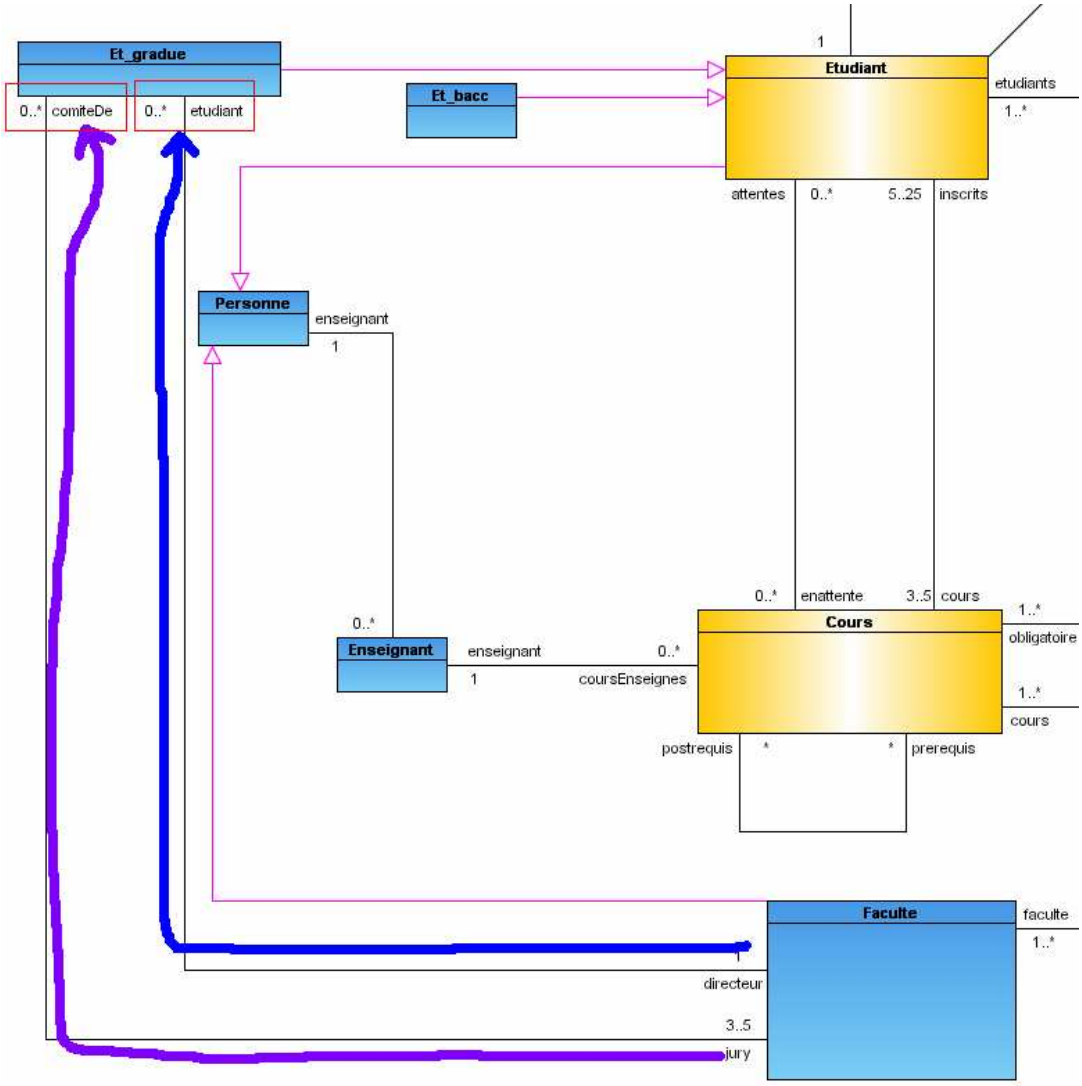

```
context i:Enseignant

inv PasEnseignementSiInscrit:

i.enseignant.oclIsKindOf(Et_gradue) implies
  i.coursEnseignes->
    intersection(i.enseignant.oclAsType(Et_gradue).cours)->
      isEmpty
```

Il ne peut y avoir d'étudiants sur la liste d'attente d'un cours si aucun étudiant n'y est inscrit




```
context c:Cours
inv NoWaitingUnlessEnrolled:
c.attentes->notEmpty implies c.inscrits->notEmpty
```

Le directeur d'un étudiant gradué est toujours sur le comité de cet étudiant



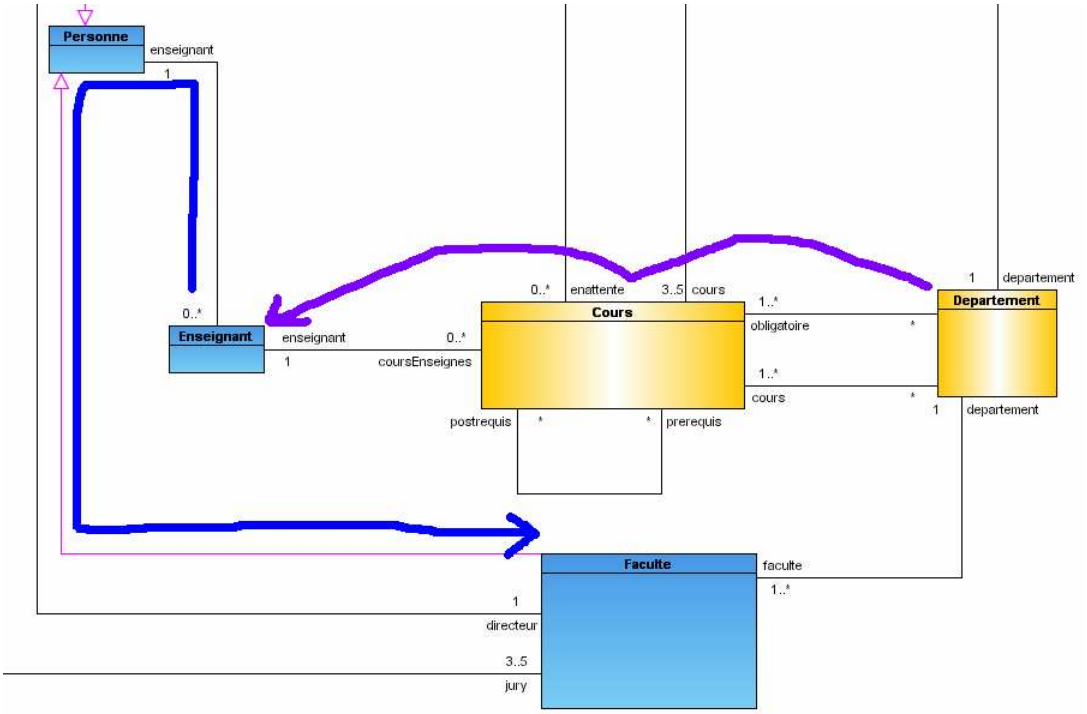

```
context f:Faculte
inv AdvisorOnCommittee:
f.comiteDe->includesAll(f.etudiant)
-- autrement dit: f.etudiant ⊆ f.comiteDe
```

Exercice :

```
context e:Et_gradue
inv DirecteurSurComite :
```

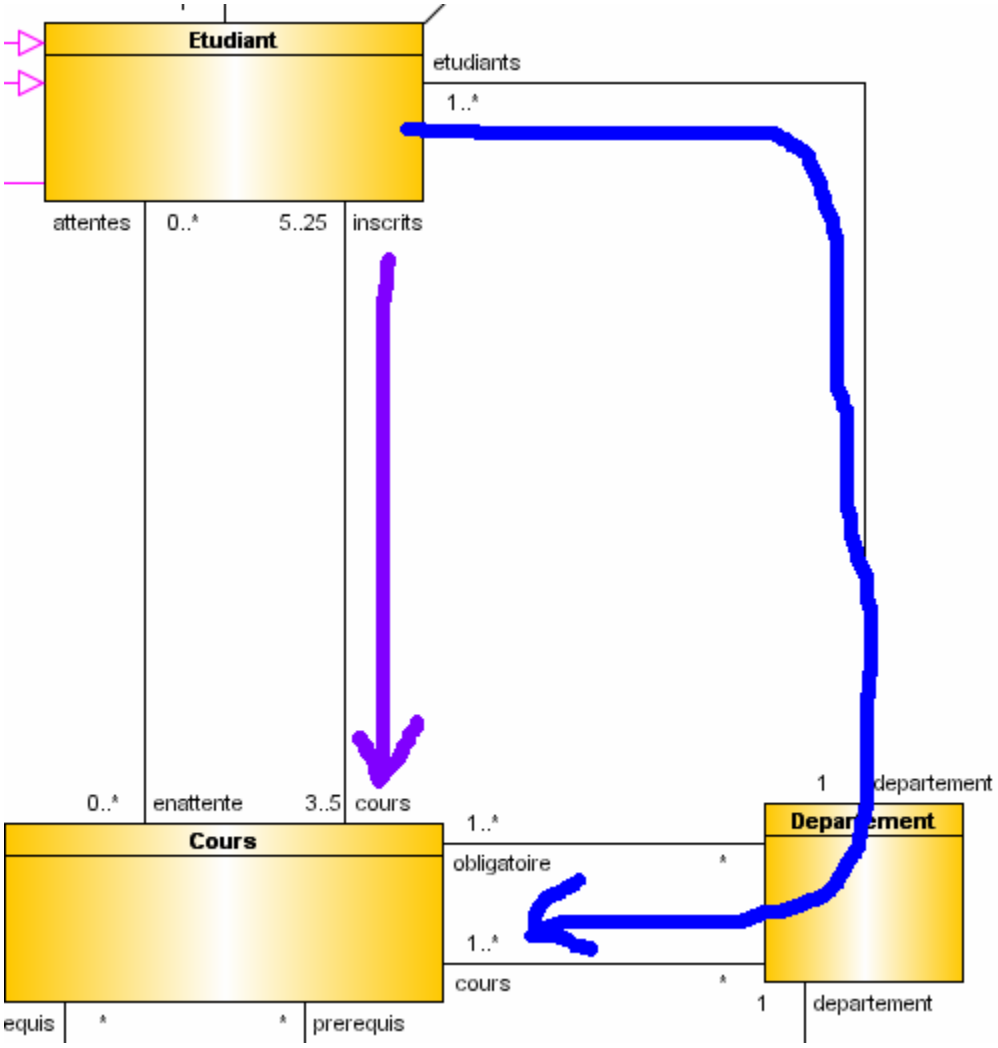
_____

Les membres du corps professoral enseignent les cours obligatoires




```
context d:Departement

inv FacultyTeachReqCourses:

d.obligatoire.enseignant->
   forAll(i:Enseignant|i.enseignant.oclIsKindOf(Faculte))
```

Les étudiants doivent prendre au moins un cours obligatoire




```
context s:Etudiant
inv OneCourseFromMajor:
s.cours->intersection(s.departement.obligatoire)->notEmpty()
```

Exercices

Exprimez des invariants pour d'autres cycles

- Les prérequis nécessaires
- Le fait de ne pas pouvoir reprendre des cours plus de trois fois (il faudra avoir recours à des attributs).