Vers la mise en oeuvre des classes

- Visibilité
- Abstraction
- Attributs et Opérations* de Classes
- Généralisation
- Packages
- Transformations des associations
- Anti-Patterns

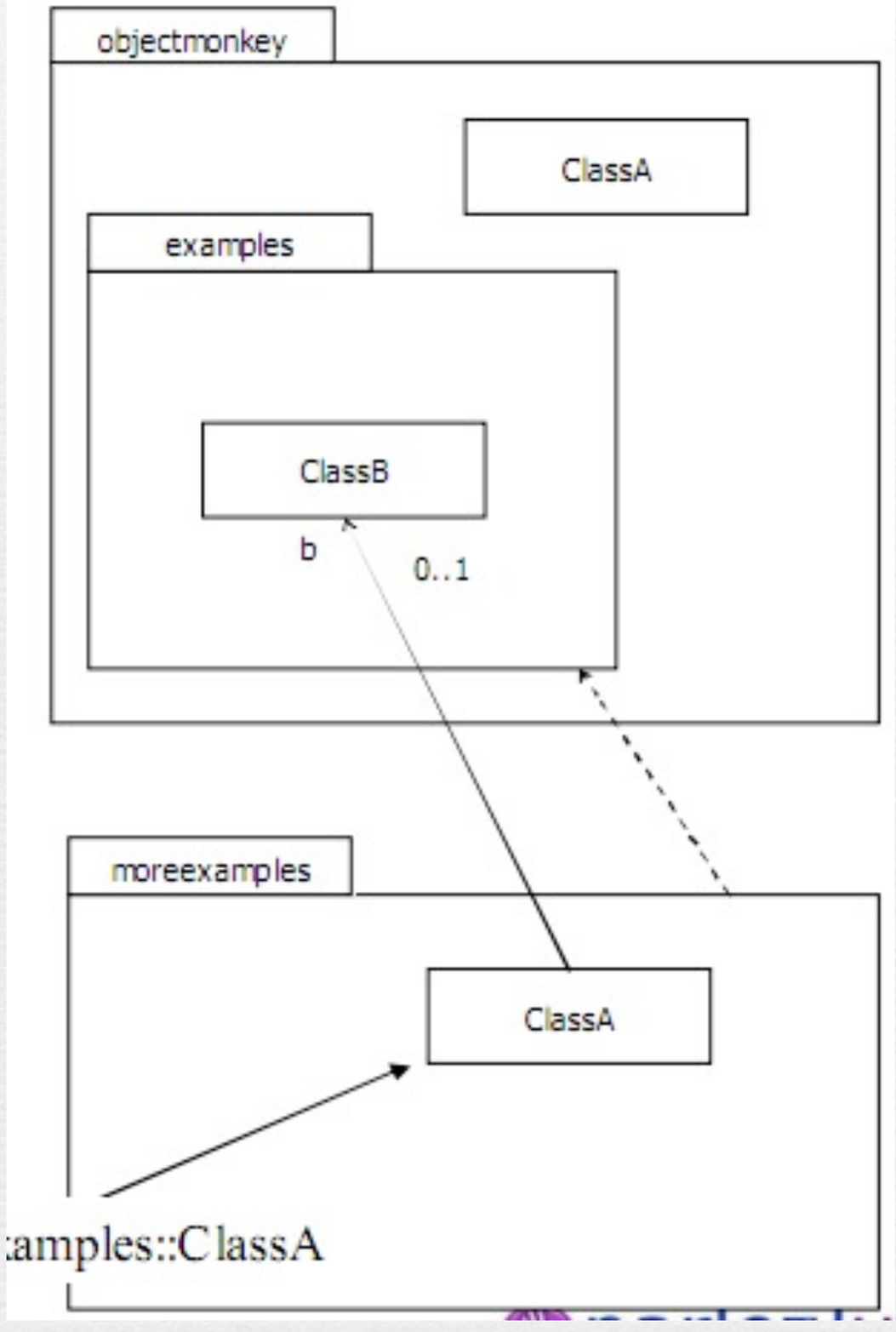
Opération : terme générique désignant le plus souvent des méthodes

Package

```
package objectmonkey;


class ClassA
{
}
```

```
package objectmonkey.examples;


class ClassB
{
}
```

```
package moreexamples;

import objectmonkey.examples.*;

class ClassA
{
        private ClassB b;
}
```

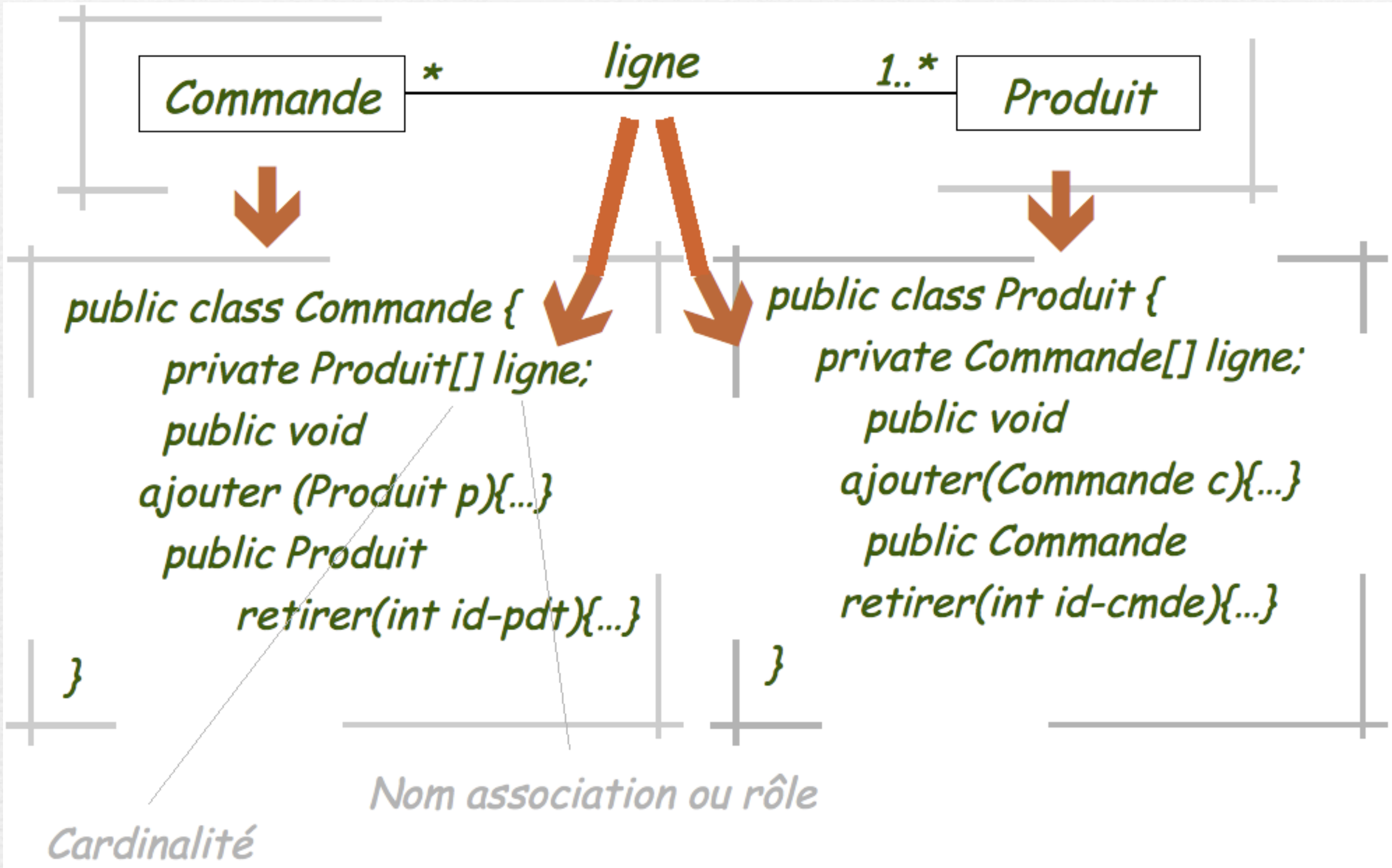


Vers la mise en oeuvre des classes

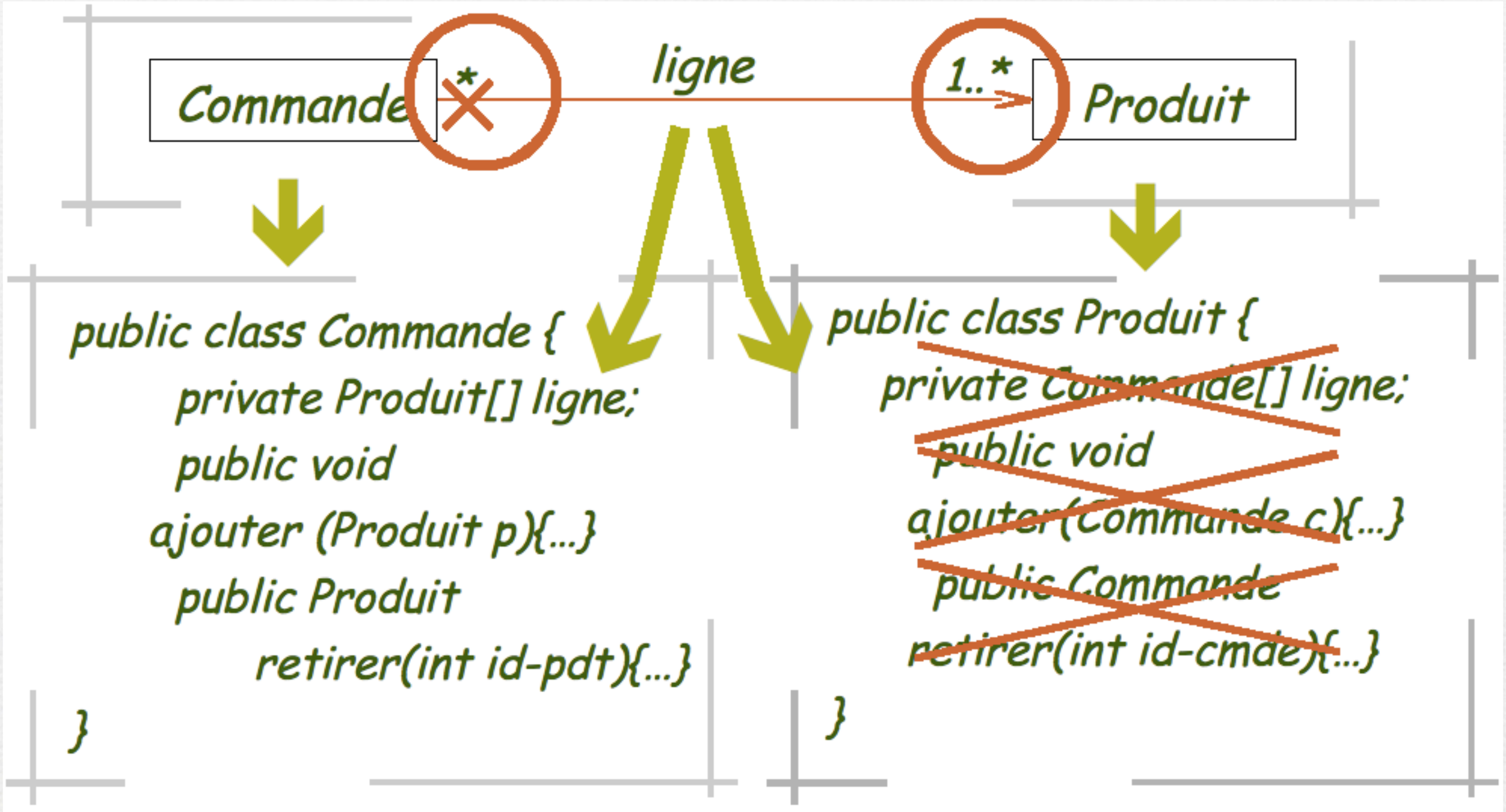
- Visibilité
- Abstraction
- Attributs et Opérations* de Classes
- Généralisation
- Packages
- Transformations des associations
- Anti-Patterns

Opération : terme générique désignant le plus souvent des méthodes

Association...



Association...



Association: De la conception à l'implémentation

```
public class Commande {
        private Produit[] ProduitsCommandés;
        public void
      ajouter (Produit p){…}
        public Produit
            retirer(int id-pdt){…}
}
```

Association:
De la conception à l'implémentation

```
public class Commande {
        private Produit[] ProduitsCommandés;
        public void
```

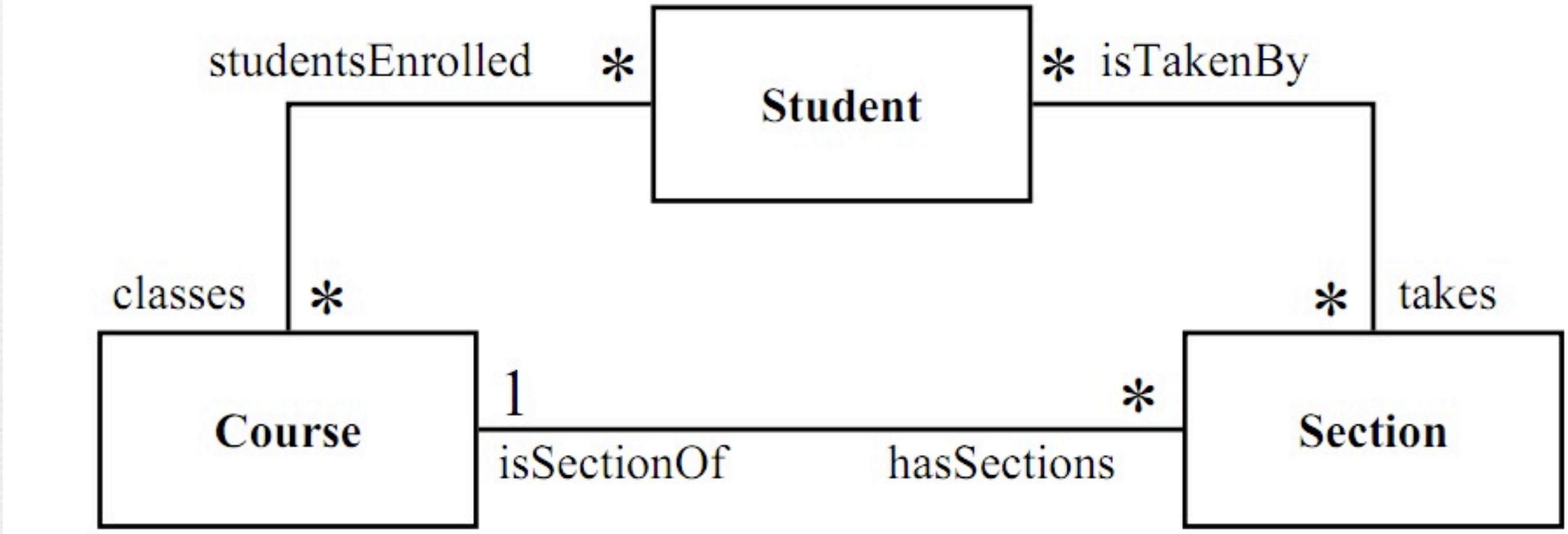
```
public Commande (Produit[] c) throws Exception {
     if (c.length != 0)
          lignes = c;
     else
          throw new Exception("Un produit au moins est requis");
}
```

Association: De la conception à l'implémentation

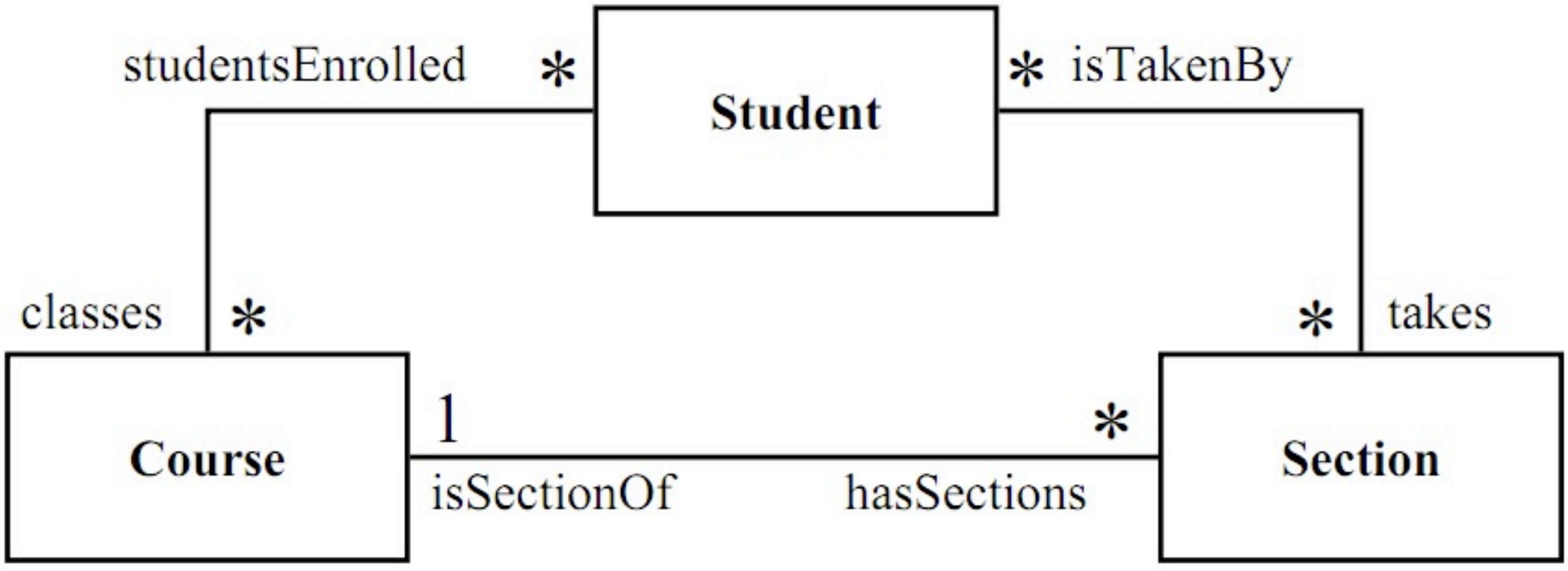
```
public Commande (Produit[] c) throws Exception {
    if (c.length != 0)
        lignes = c;
    else
        throw new Exception("Un produit au moins est
requis");
    }
```

```
    public boolean oterProduit(Course c) {
        if (lignes.length==1)
            return false;
        ...
    }
```

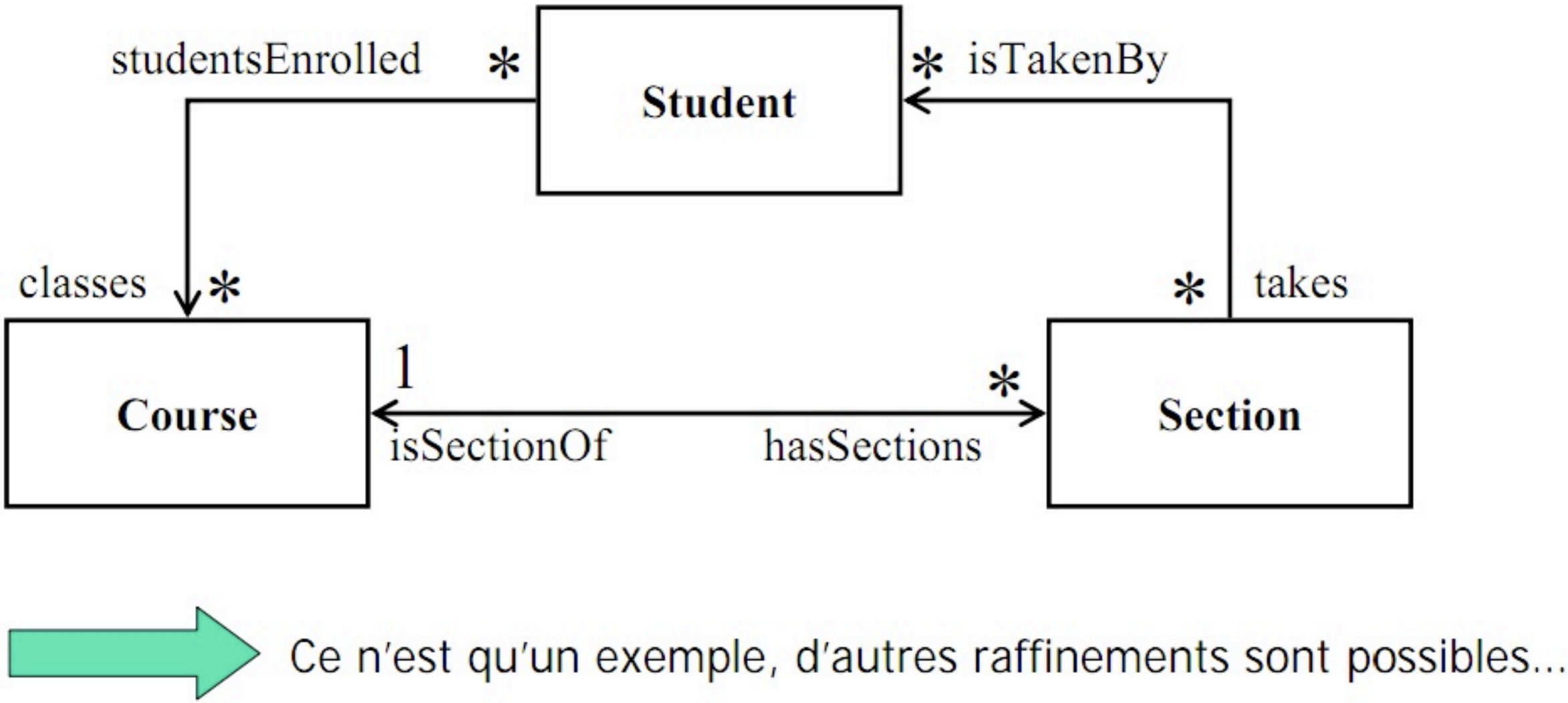
Gestion des associations



Associations & Navigations



Exemple de Raffinement



Ce n'est qu'un exemple, d'autres raffinements sont possibles...

Principes d'implémentation

Principes d'implémentation

- Extrémité d'association 1

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *
  - Rôle (pluriel) en collection

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *
  - Rôle (pluriel) en collection
  - Type de l'extrémité en élément de collection

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *
  - Rôle (pluriel) en collection
  - Type de l'extrémité en élément de collection
  - Collection get*Roles*()

Principes d'implémentation

- Extrémité d'association 1
  - Rôle en Attribut avec type de l'extrémité
  - *Type* get*Role*()
- Extrémité d'association *
  - Rôle (pluriel) en collection
  - Type de l'extrémité en élément de collection
  - Collection get*Roles*()
  - // Collection<TypeExtrémité>//

Principes d'implémentation (Suite)

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - void setRole(Type t)

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - void setRole(Type t)
- *Fixer* une extrémité d'association *

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - void setRole(Type t)
- *Fixer* une extrémité d'association *
  - void setRoles(Collection c)

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - void setRole(Type t)
- *Fixer* une extrémité d'association *
  - void setRoles(Collection c)
  - void addRole(TypeElement t)

Principes d'implémentation (Suite)

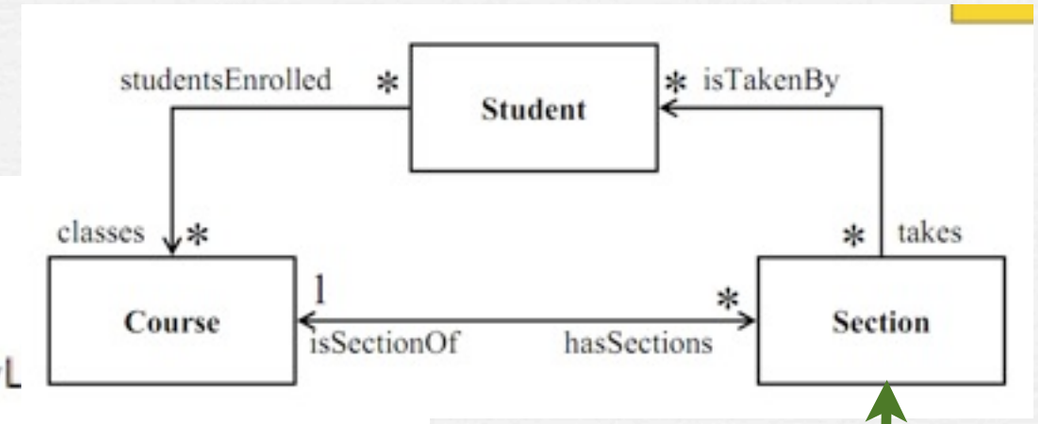
- *Fixer* une extrémité d'association 1
  - void setRole(Type t)
- *Fixer* une extrémité d'association *
  - void setRoles(Collection c)
  - void addRole(TypeElement t)
- *Fixer* une association navigable dans les 2 sens :

Principes d'implémentation (Suite)

- *Fixer* une extrémité d'association 1
  - void setRole(Type t)
- *Fixer* une extrémité d'association *
  - void setRoles(Collection c)
  - void addRole(TypeElement t)
- *Fixer* une association navigable dans les 2 sens :
  - Définir les responsabilités : un des objets est responsable de la connexion/déconnexion (cf. exemple)

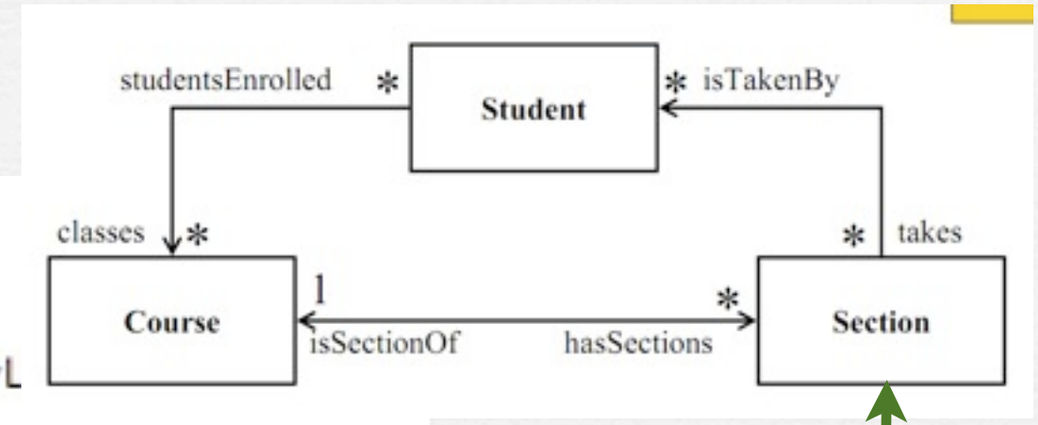
Implémentation

```
public class Section {
    private String name;
    private Course isSectionOf;
    private  Collection<Student> isTakenBy = new ArrayL
    public String toString() {
    public Section(String name, Course isSectionOf) {
        this.name = name;
        //ATTENTION ...
        setIsSectionOf(isSectionOf) ;
    }

    public void setIsSectionOf(Course c) {
        isSectionOf = c;
        c.addHasSections(this);
    }

    public Course getIsSectionOf() { return isSectionOf;}

    public Collection<Student> getIsTakenBy() {
        return isTakenBy;
    }

    public void addStudent(Student s) {
        isTakenBy.add(s);
        if (!(s.getClasses().contains(isSectionOf)) )
            s.addClass(isSectionOf);
    }
```



Prise de responsabilités

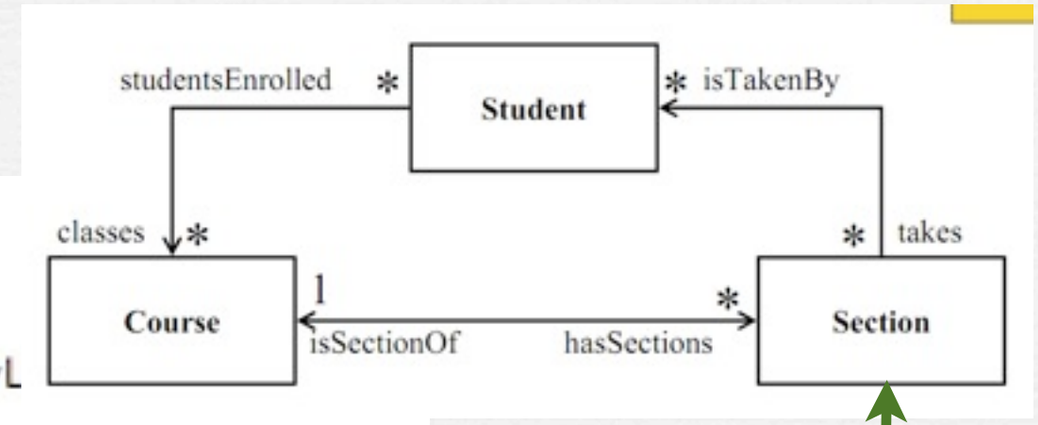
Implémentation




```
public class Section {
    private String name;
    private Course isSectionOf;
    private  Collection<Student> isTakenBy = new ArrayL
    public String toString() {
    public Section(String name, Course isSectionOf) {
        this.name = name;
        //ATTENTION ...
        setIsSectionOf(isSectionOf) ;
    }

    public void setIsSectionOf(Course c) {
        isSectionOf = c;
        c.addHasSections(this);
    }

    public Course getIsSectionOf() { return isSectionOf;}

    public Collection<Student> getIsTakenBy() {
        return isTakenBy;
    }

    public void addStudent(Student s) {
        isTakenBy.add(s);
        if (!(s.getClasses().contains(isSectionOf)) )
            s.addClass(isSectionOf);
    }
```

Prise de responsabilités

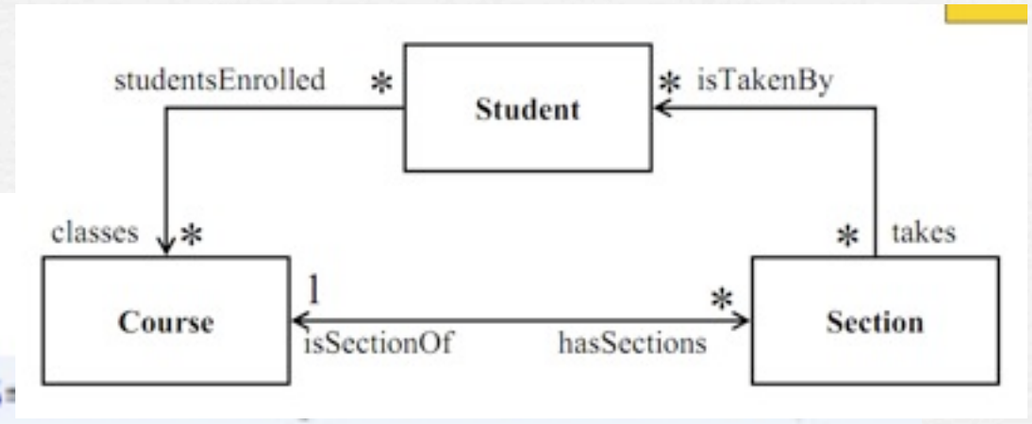
Implémentation




```
public class Section {
    private String name;
    private Course isSectionOf;
    private  Collection<Student> isTakenBy = new ArrayL
    public String toString() {
    public Section(String name, Course isSectionOf) {
        this.name = name;
        //ATTENTION ...
        setIsSectionOf(isSectionOf) ;
    }

    public void setIsSectionOf(Course c) {
        isSectionOf = c;
        c.addHasSections(this);
    }

    public Course getIsSectionOf() { return isSectionOf;}

    public Collection<Student> getIsTakenBy() {
        return isTakenBy;
    }

    public void addStudent(Student s) {
        isTakenBy.add(s);
        if (!(s.getClasses().contains(isSectionOf)) )
            s.addClass(isSectionOf);
    }
```

Prise de responsabilités

Implémentation

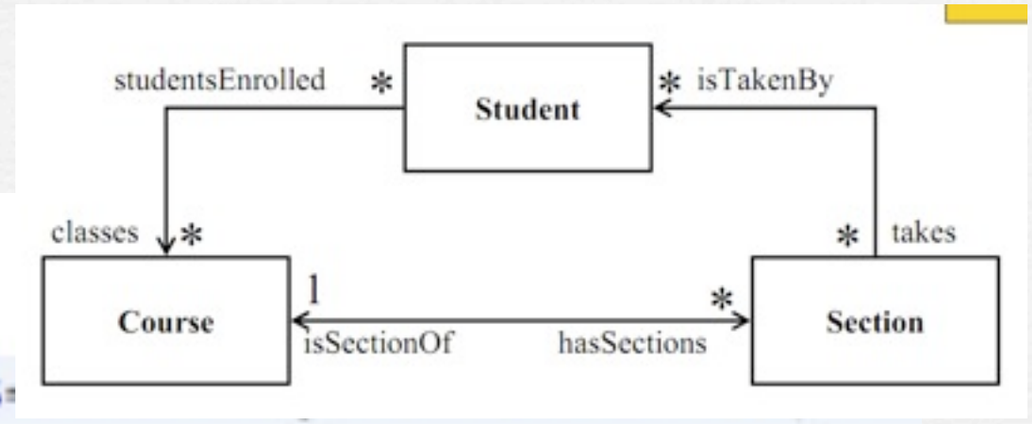



```
public class Student {
        private String name;
        private Collection<Course> classes=

        public Student(String name) {..
        public String toString() {..
        public Collection<Course> getClasses() {
                return classes;
            }
        protected void addClass(Course c){
                classes.add(c);
```

Définition des responsabilités
Ne jamais appeler ***addHasSections*** ou ***addClass*** directement !

```
public class Course {
    private String name;
    private Collection<Section>  hasSections = new ArrayList<Section>();

    public Course(String name) {..
    public String toString() {..
    public Collection<Section> getHasSections() {
        return hasSections;}
    protected void addHasSections(Section s){
                hasSections.add(s);
            }
```

Implémentation




```
public class Student {
        private String name;
        private Collection<Course> classes
```

```
    public Student(String name) {..
    public String toString() {..
    public Collection<Course> getClasses() {
        return classes;
    }
```

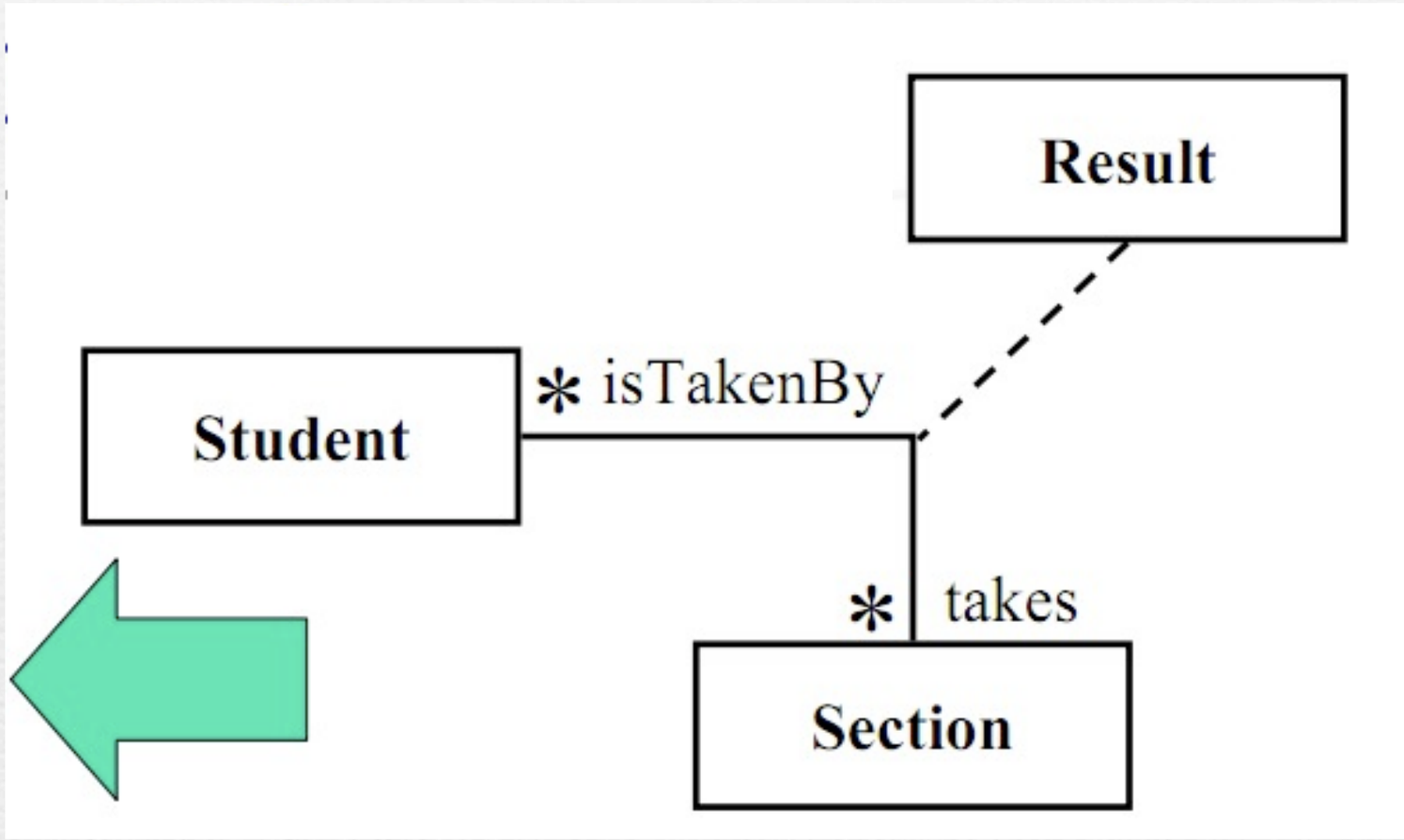
```
    protected void addClass(Course c){
        classes.add(c);
```

Définition des responsabilités
Ne jamais appeler ***addHasSections*** ou ***addClass*** directement !

```
public class Course {
    private String name;
    private Collection<Section>  hasSections = new ArrayList<Section>();

    public Course(String name) {..
    public String toString() {..
    public Collection<Section> getHasSections() {
        return hasSections;}
    protected void addHasSections(Section s){
            hasSections.add(s);
        }
```

Implémentation

```
class Student {
   Result getResult(Section s)
  }
class Section {
  Result getResult(Student s)
 }
class Result {
  Student getStudent()
  Section getSection()
}
```



En résumé : Traduction des associations en attributs

En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)

En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche

En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche
- Nom de l'attribut = nom du rôle ou forme nominale du nom de l'association

En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche
- Nom de l'attribut = nom du rôle ou forme nominale du nom de l'association
- Attribut du type référence sur un objet de la classe à l'autre extrémité de l'association
  - Référence notée « @ »

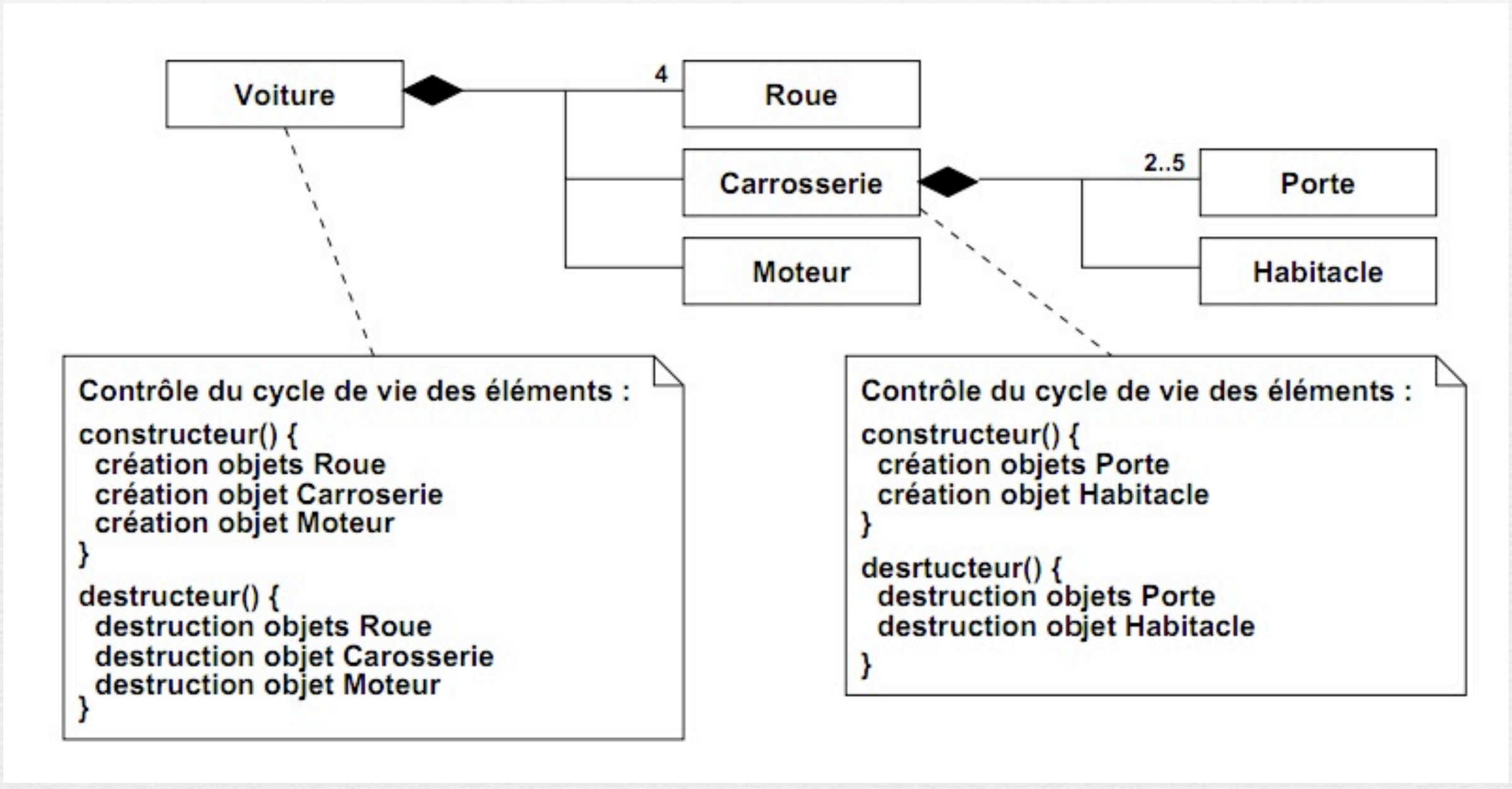
En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche
- Nom de l'attribut = nom du rôle ou forme nominale du nom de l'association
- Attribut du type référence sur un objet de la classe à l'autre extrémité de l'association
  - Référence notée « @ »
- Traduction des multiplicités
  - 1 =⇒ @Classe
  - * =⇒ Collection @Classe
  - 0..N =⇒ Tableau[N] Classe

En résumé : Traduction des associations en attributs

- Autant d'attributs que de classes auxquelles elle est reliée (navigable)
- Association unidirectionnelle = pas d'attribut du côté de la flèche
- Nom de l'attribut = nom du rôle ou forme nominale du nom de l'association
- Attribut du type référence sur un objet de la classe à l'autre extrémité de l'association
  - Référence notée « @ »
- Traduction des multiplicités
  - 1 =⇒ @Classe
  - * =⇒ Collection @Classe
  - 0..N =⇒ Tableau[N] Classe
- Multiplicité avec tri = Collection ordonnée @Classe

Compositions



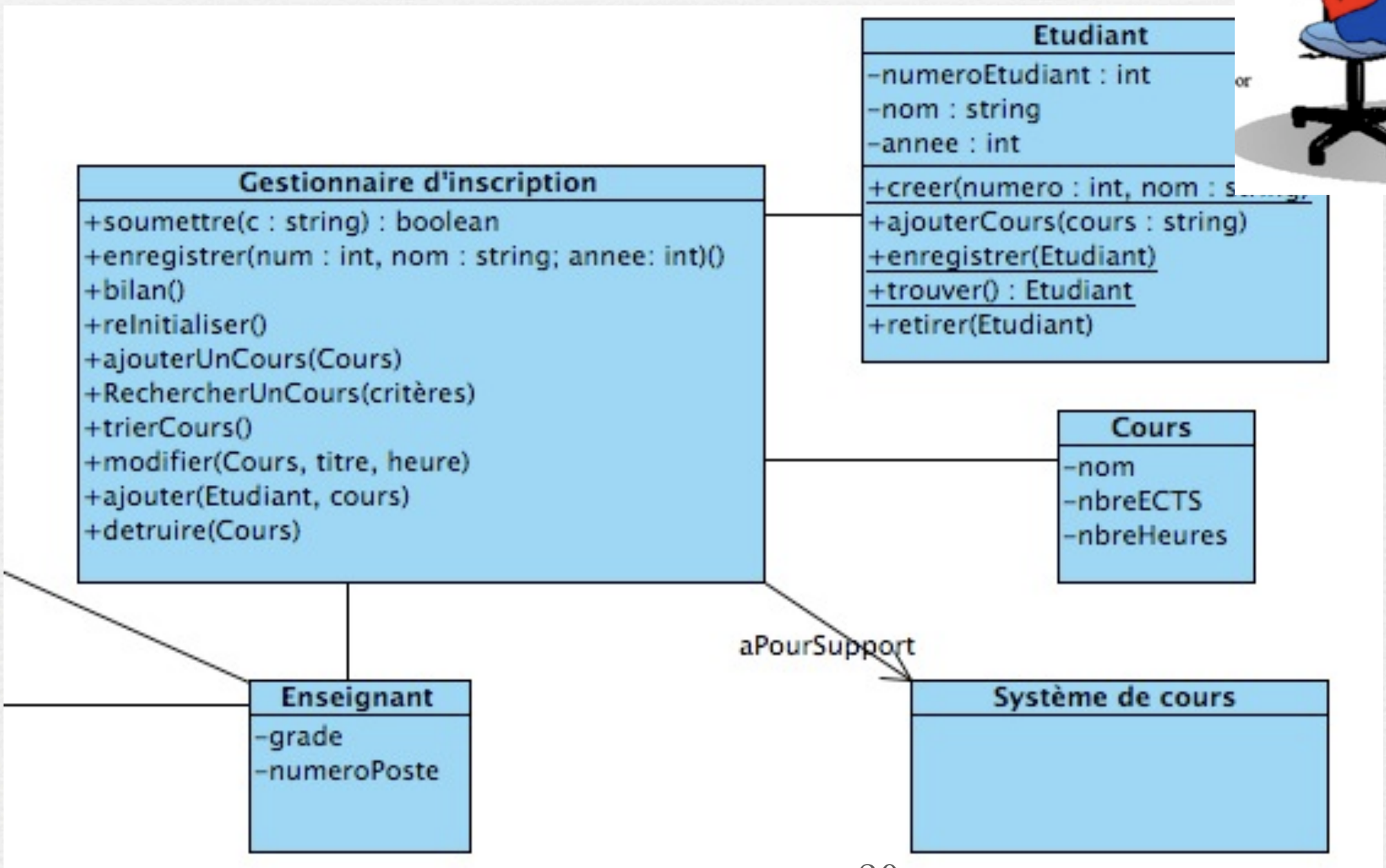
Vers la mise en oeuvre des classes

- Visibilité
- Abstraction
- Attributs et Opérations* de Classes
- Généralisation
- Packages
- Transformations des associations
- Anti-Patterns

Opération : terme générique désignant le plus souvent des méthodes

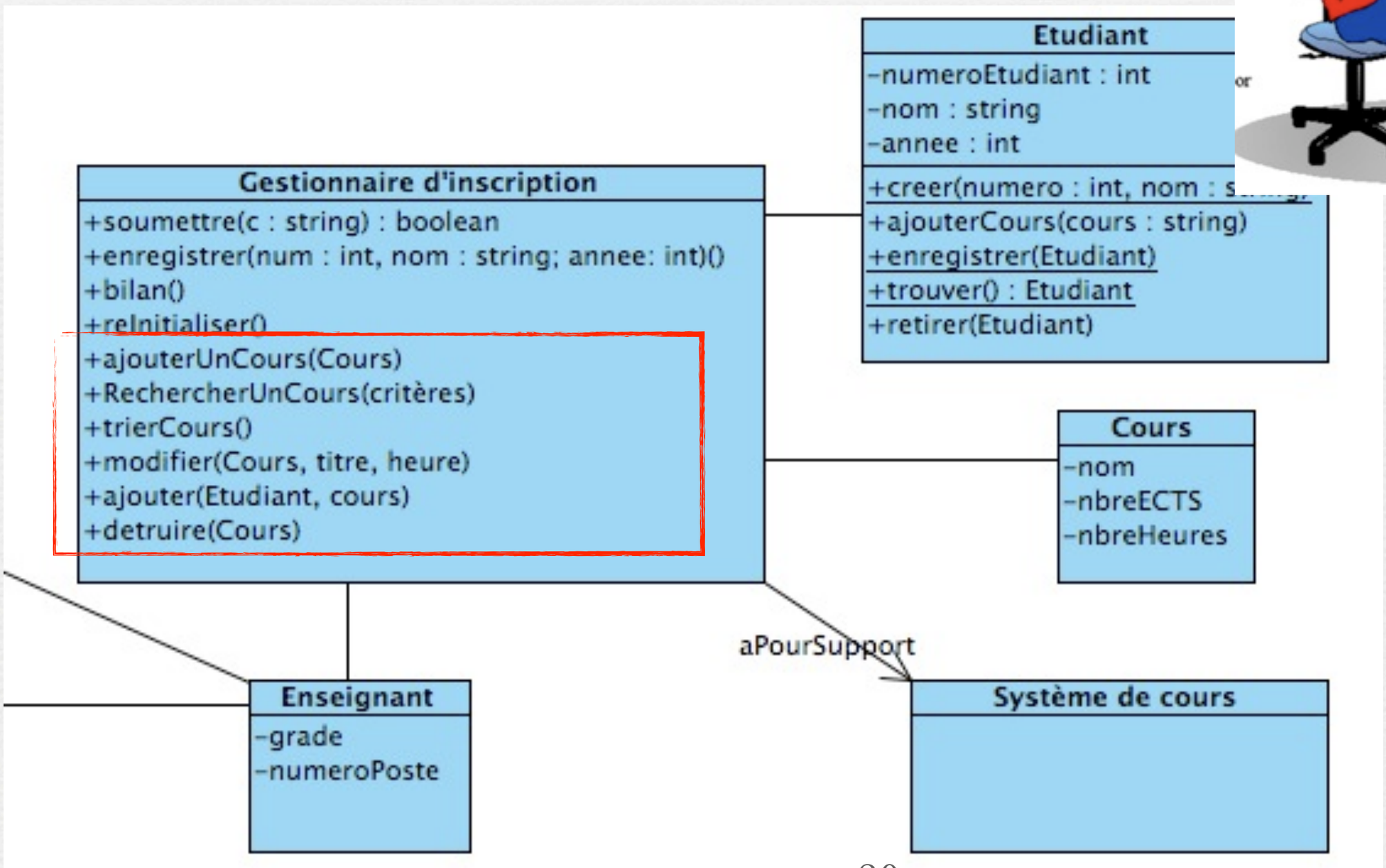
Anti-Patterns

Blob ou la classe Dieu



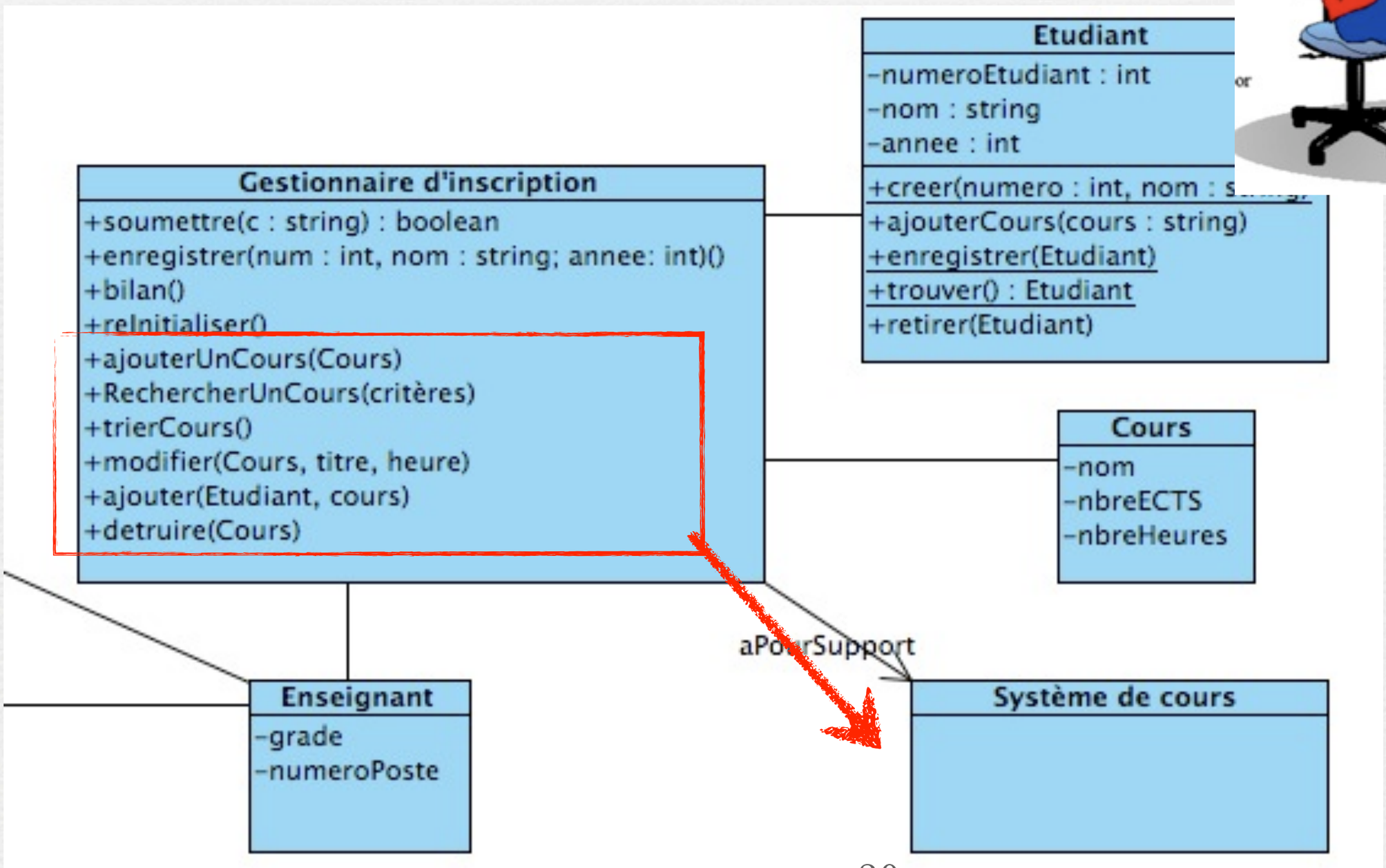
Anti-Patterns

Blob ou la classe Dieu



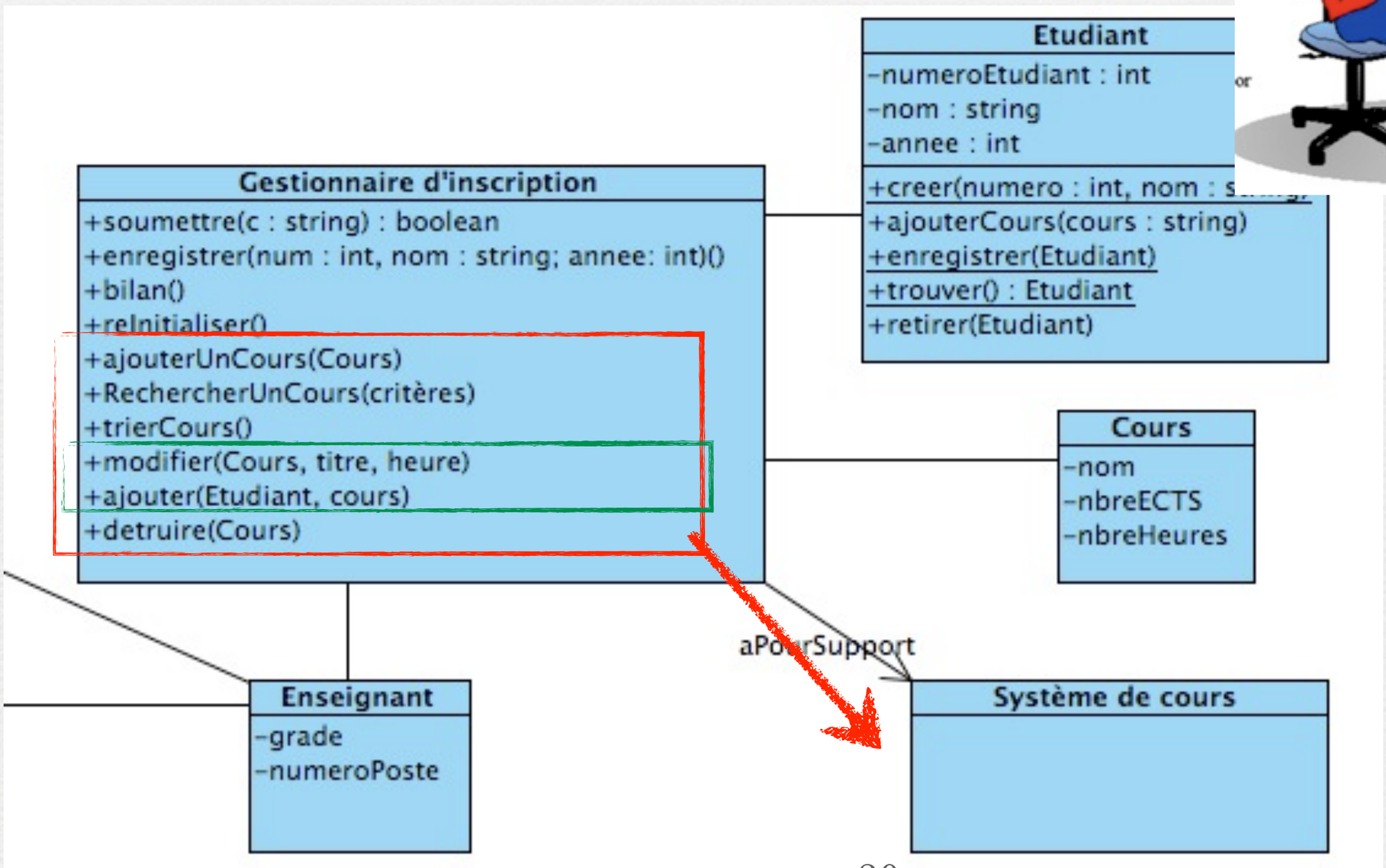
Anti-Patterns

Blob ou la classe Dieu



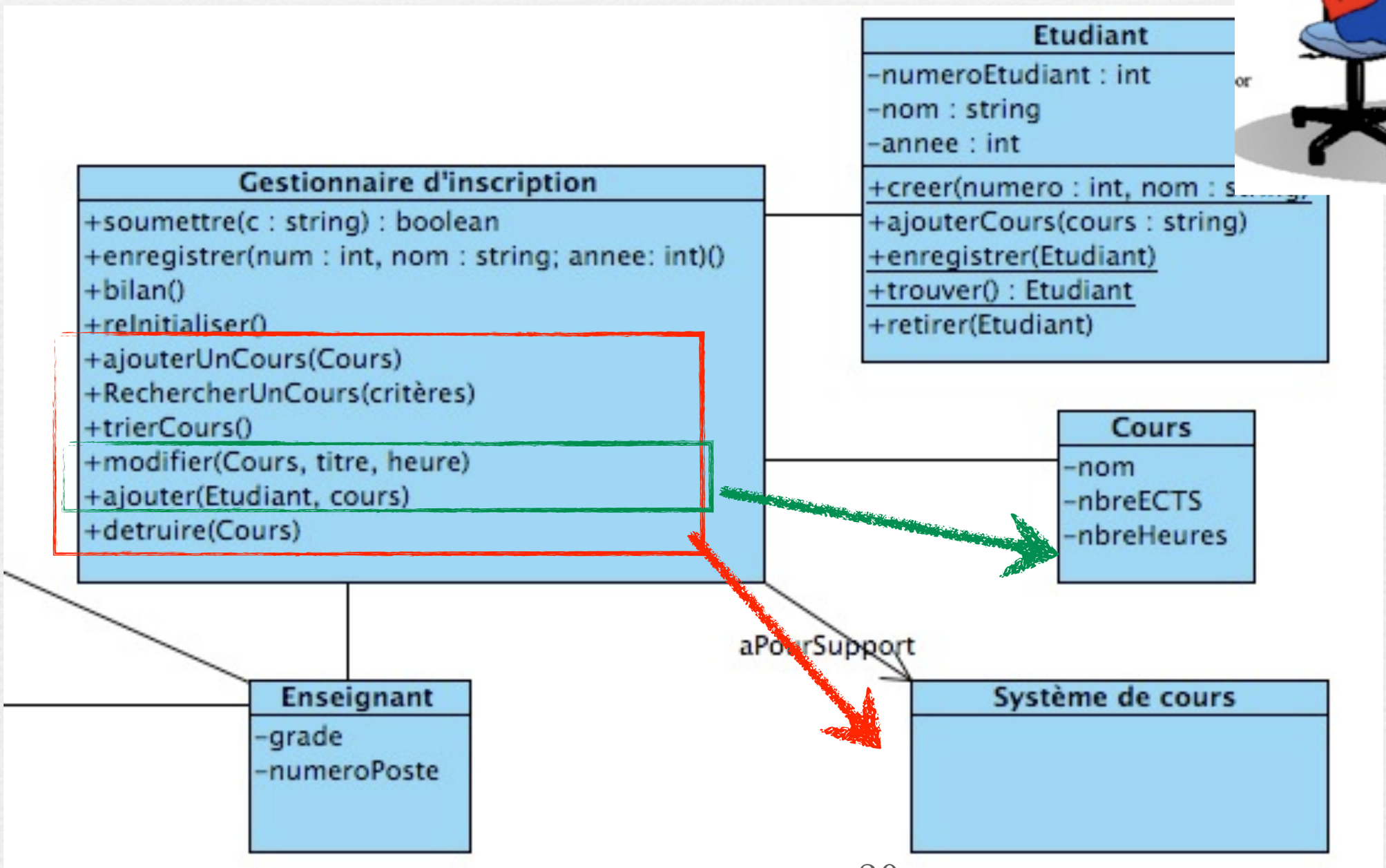
Anti-Patterns

Blob ou la classe Dieu



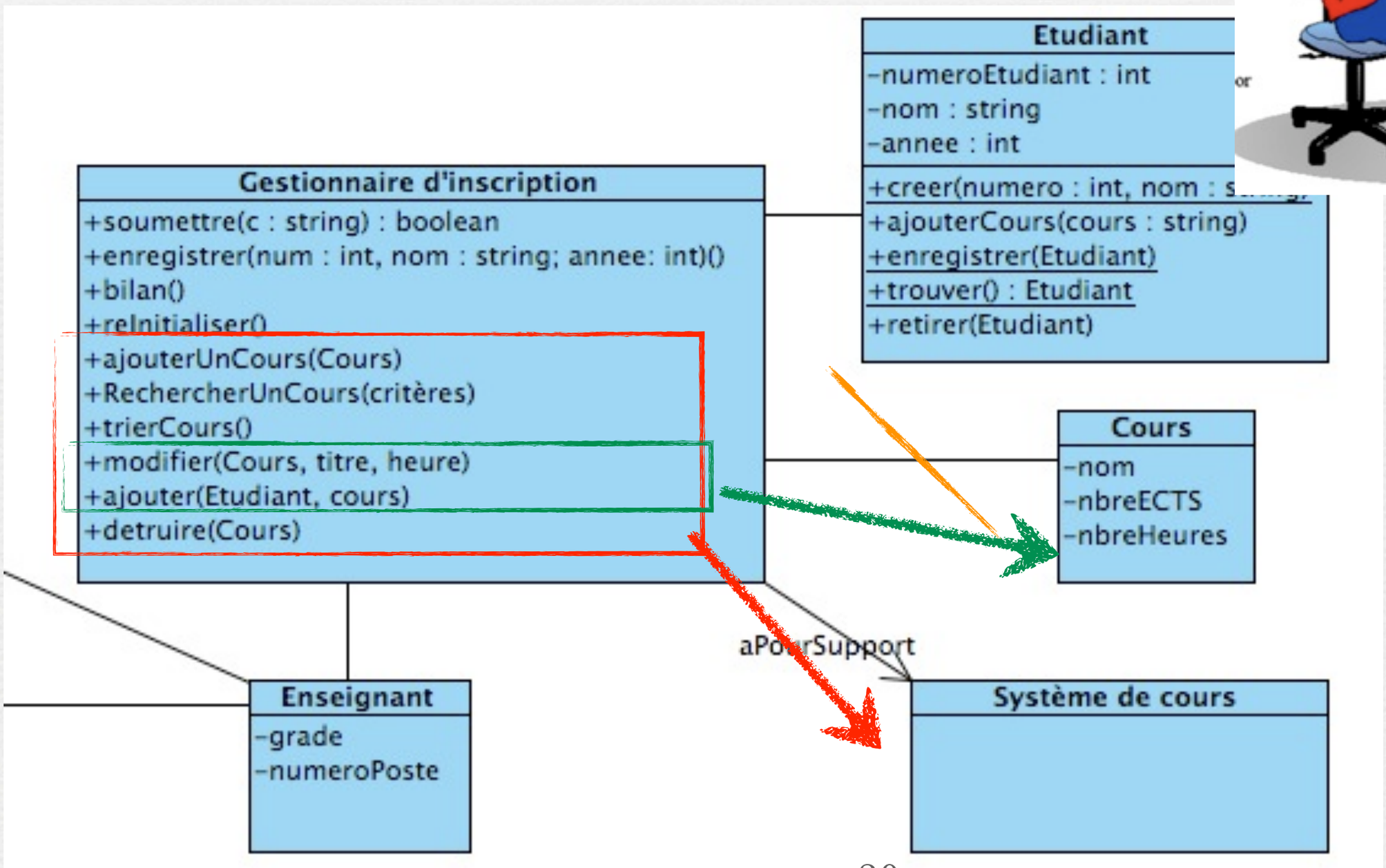
Anti-Patterns

Blob ou la classe Dieu

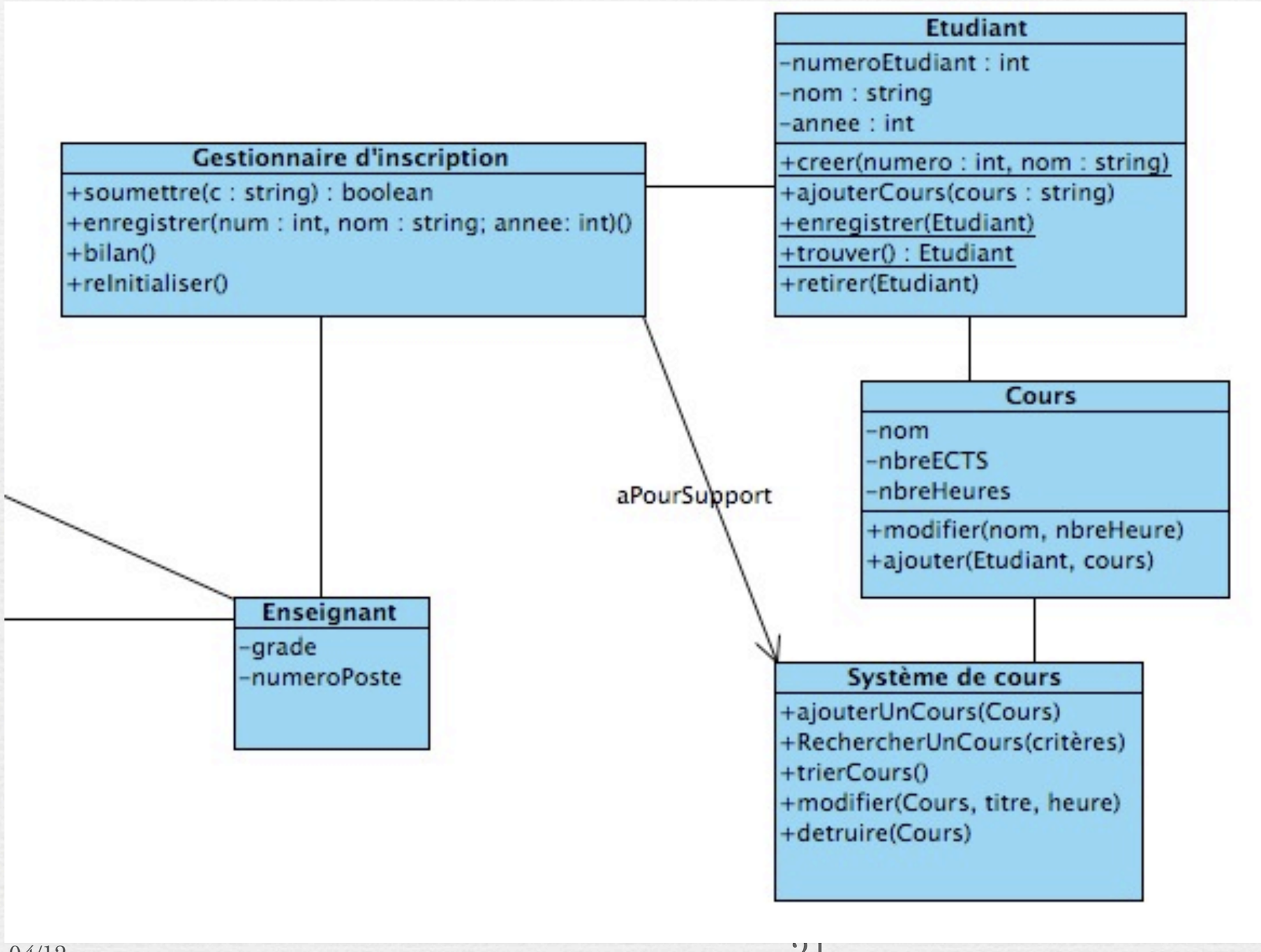


Anti-Patterns

Blob ou la classe Dieu



Anti-Patterns



A Améliorer encore...

D'un modèle à classes au modèle relationnel