Analyse et Conception avec UML

De l'analyse à la conception détaillée : - partie 2 - Passage en conception

blay@unice.fr
www.polytech.unice.fr/~blay
IUT Nice-Sophia Antipolis
avril 2012
Site web du module : http://anubis.polytech.unice.fr/iut/

Quizz

QUIZZ

Combien d'instance de H sont associées à E ?

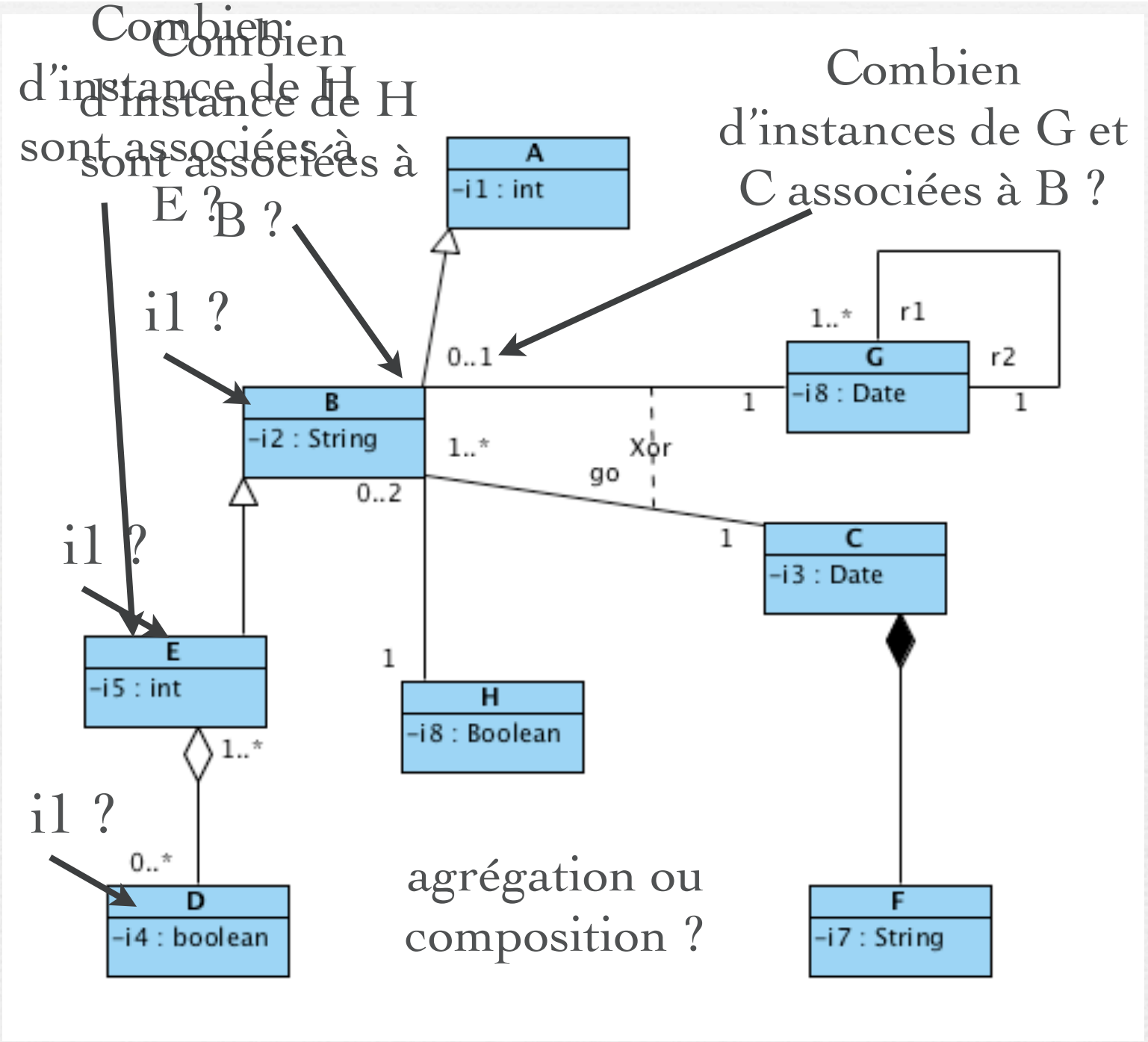
Combien d'instance de H sont associées à B ?

Combien d'instances de G et C associées à B ?

i1 ?

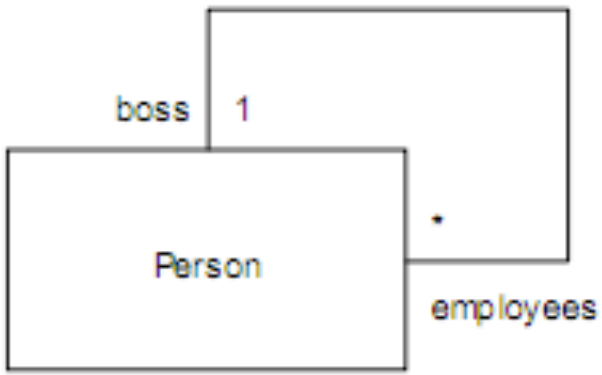
i1 ?

i1 ?



agrégation ou composition ?

QUIZZ



Which of the following sentences can apply to the class model?

a) "Be your own boss"
b) "Neither a leader nor a follower be"
c) "Sitting at the top of the tree"
d) "Too many chiefs, not enough Indians"

De Analyse à la conception

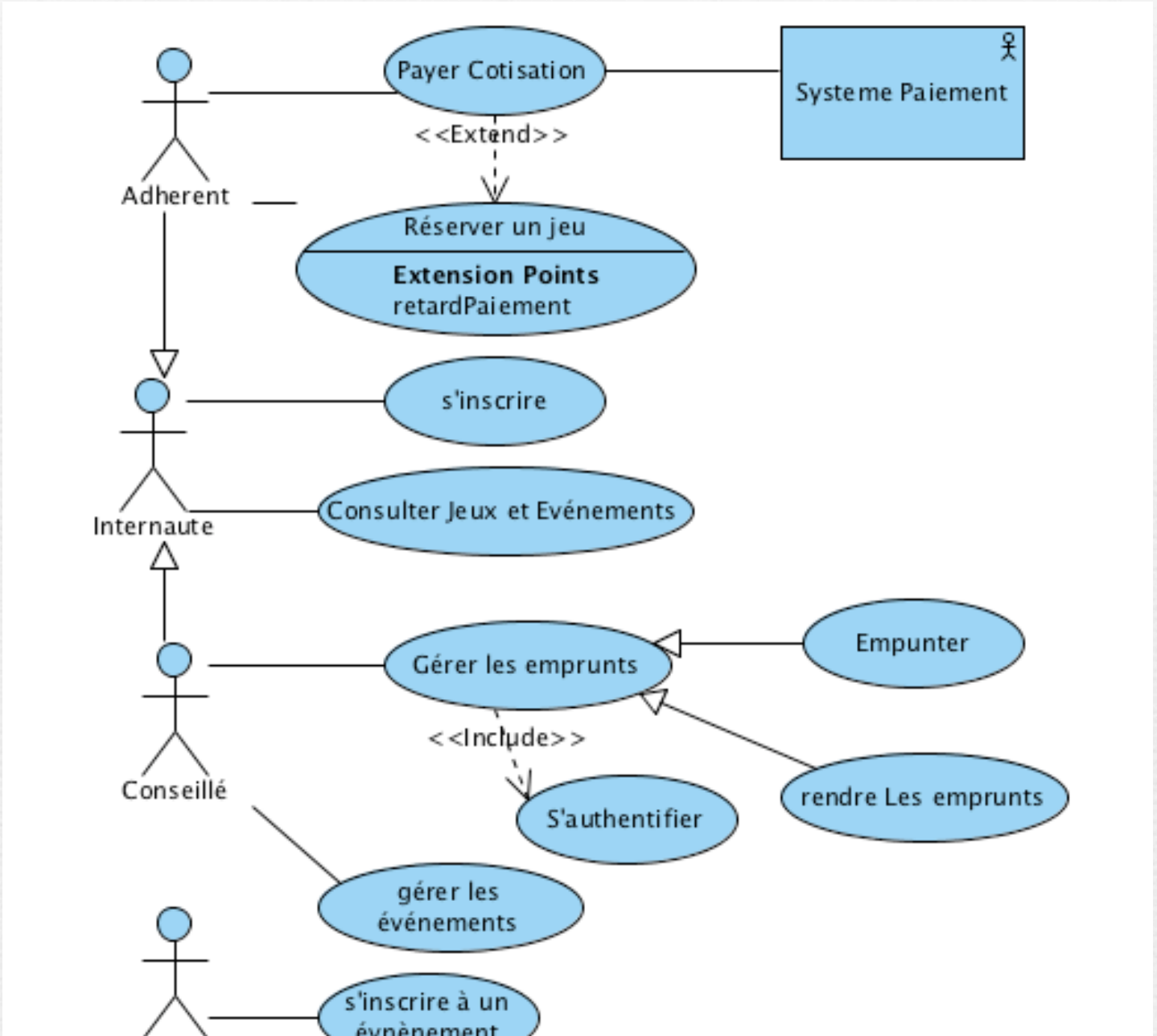
- Pattern exemplaire
- Structuration en package
- Structuration en package et réutilisation

- Choix des itérations
- Diag. de séquence en conception
- Architecture
- Des diag. de séquences aux Diag. de classes
- Diagrammes de classes en conception

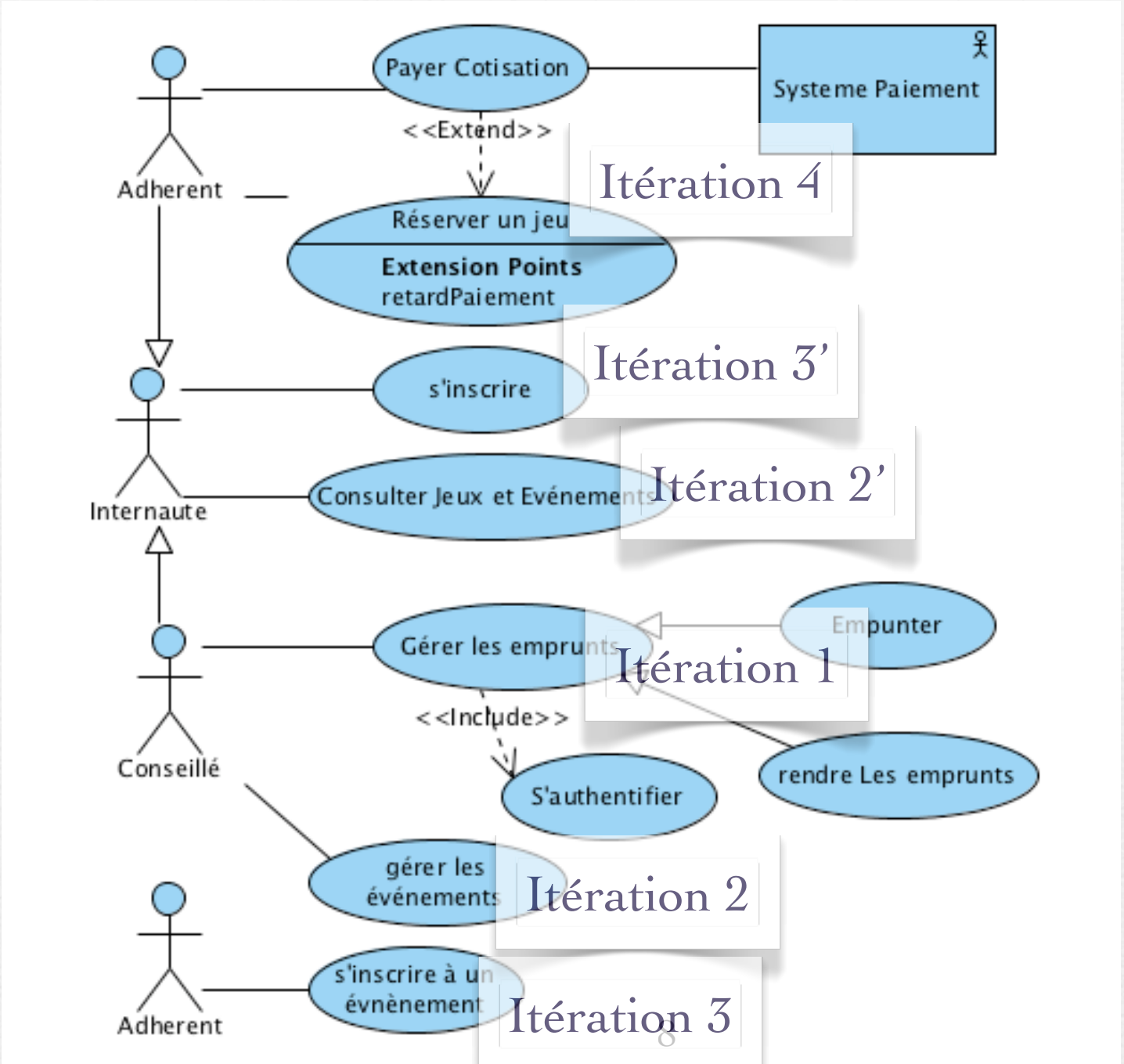
Vers la conception : choix des itérations

Choix des itérations

On a

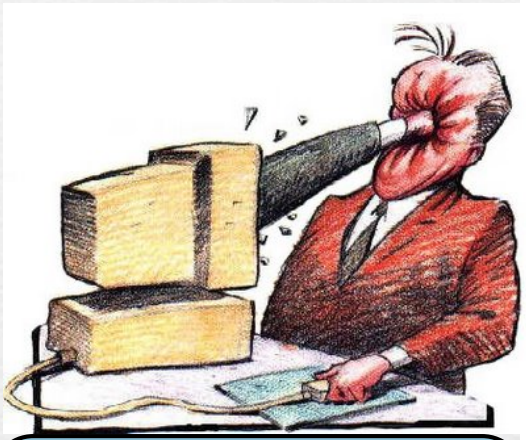


Choix des itérations

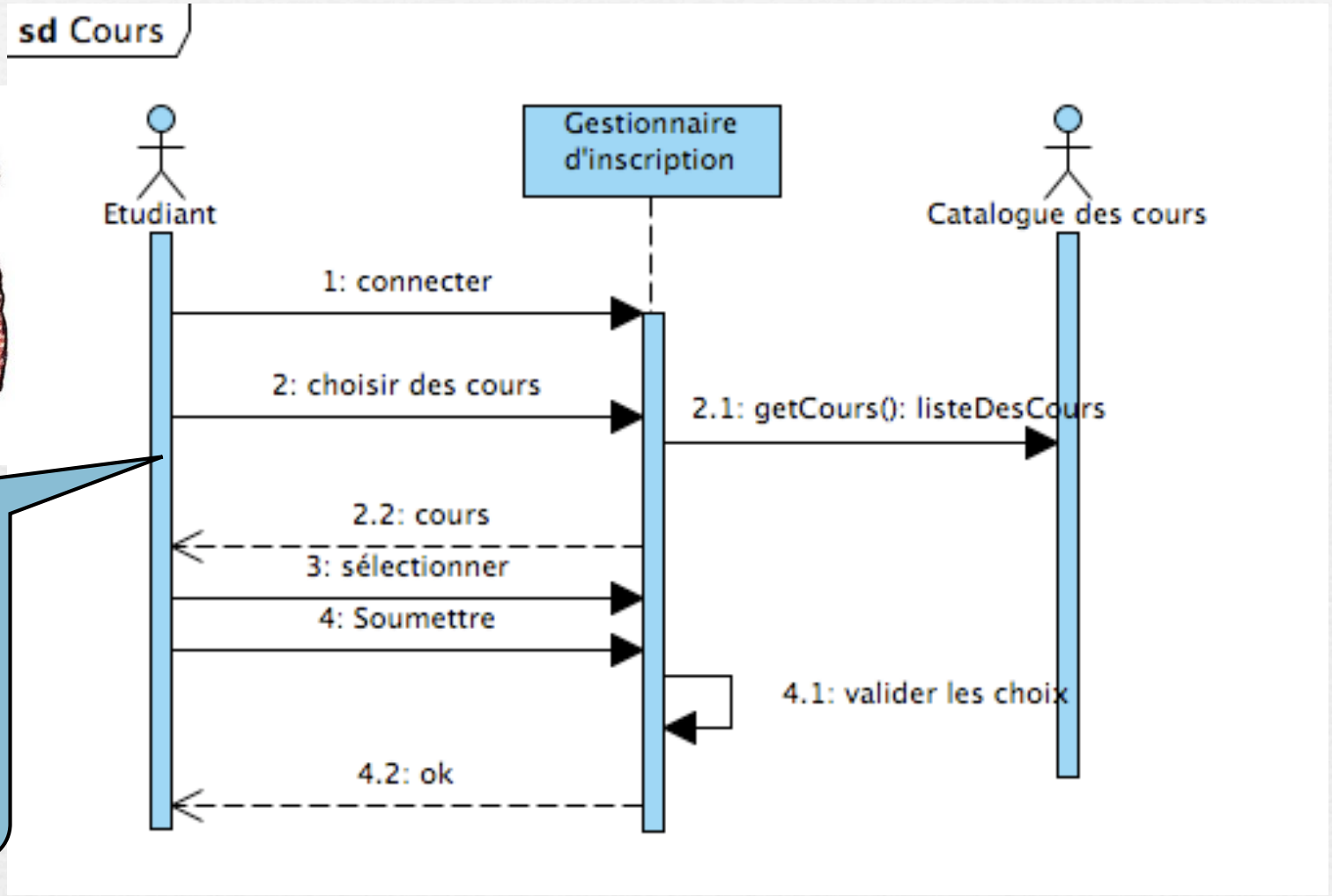


Des diagrammes de séquence en conception

Vers l'architecture



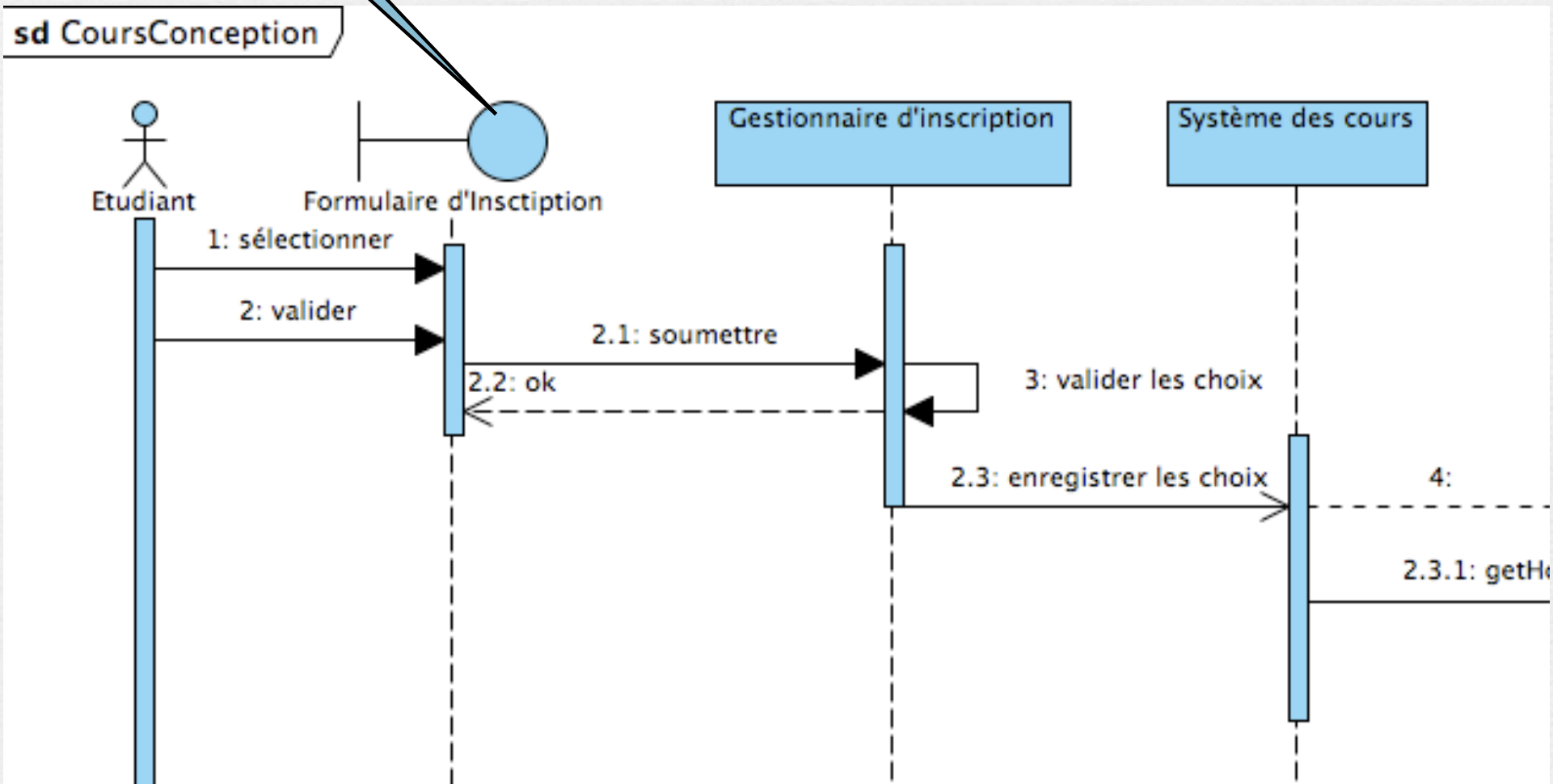
Pas d'interaction directe du client!



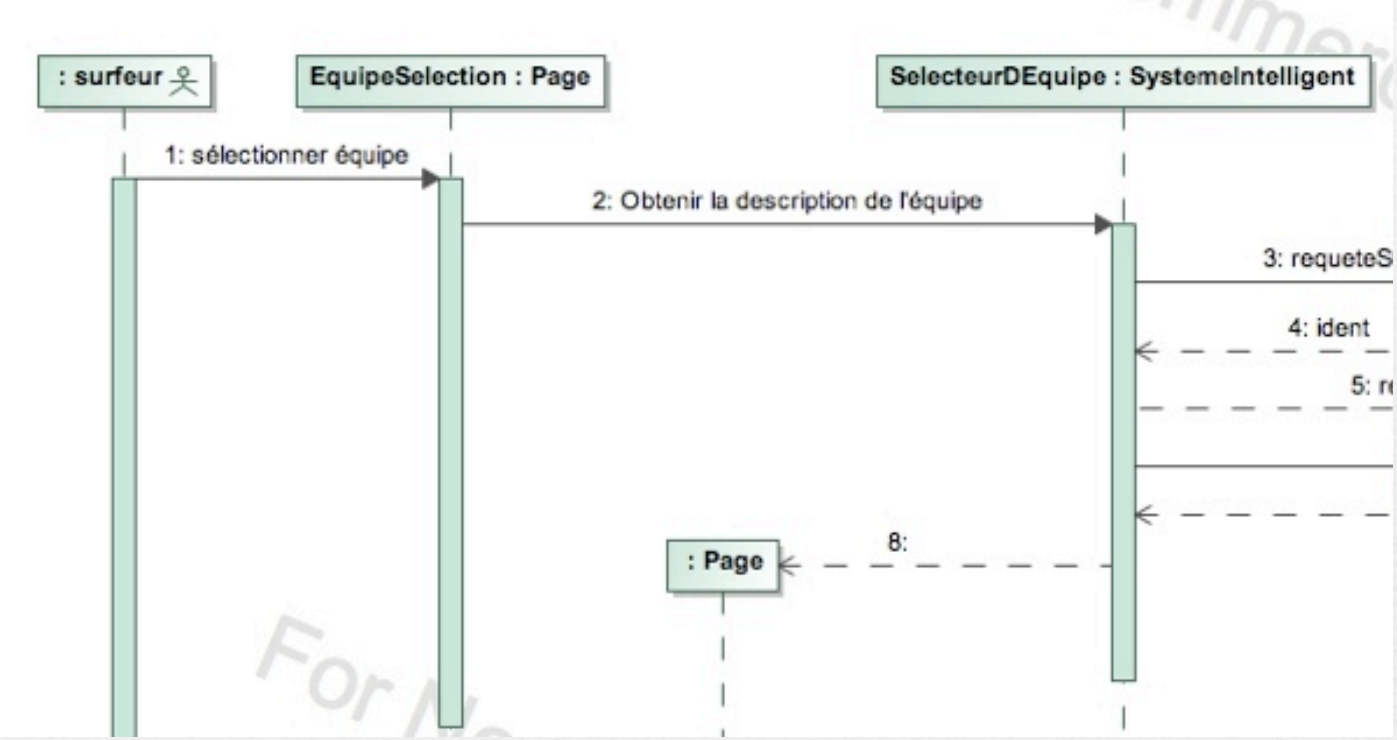
Les "interfaces avec les acteurs" sont «réifiées».

Vers l'architecture

Objet Frontière



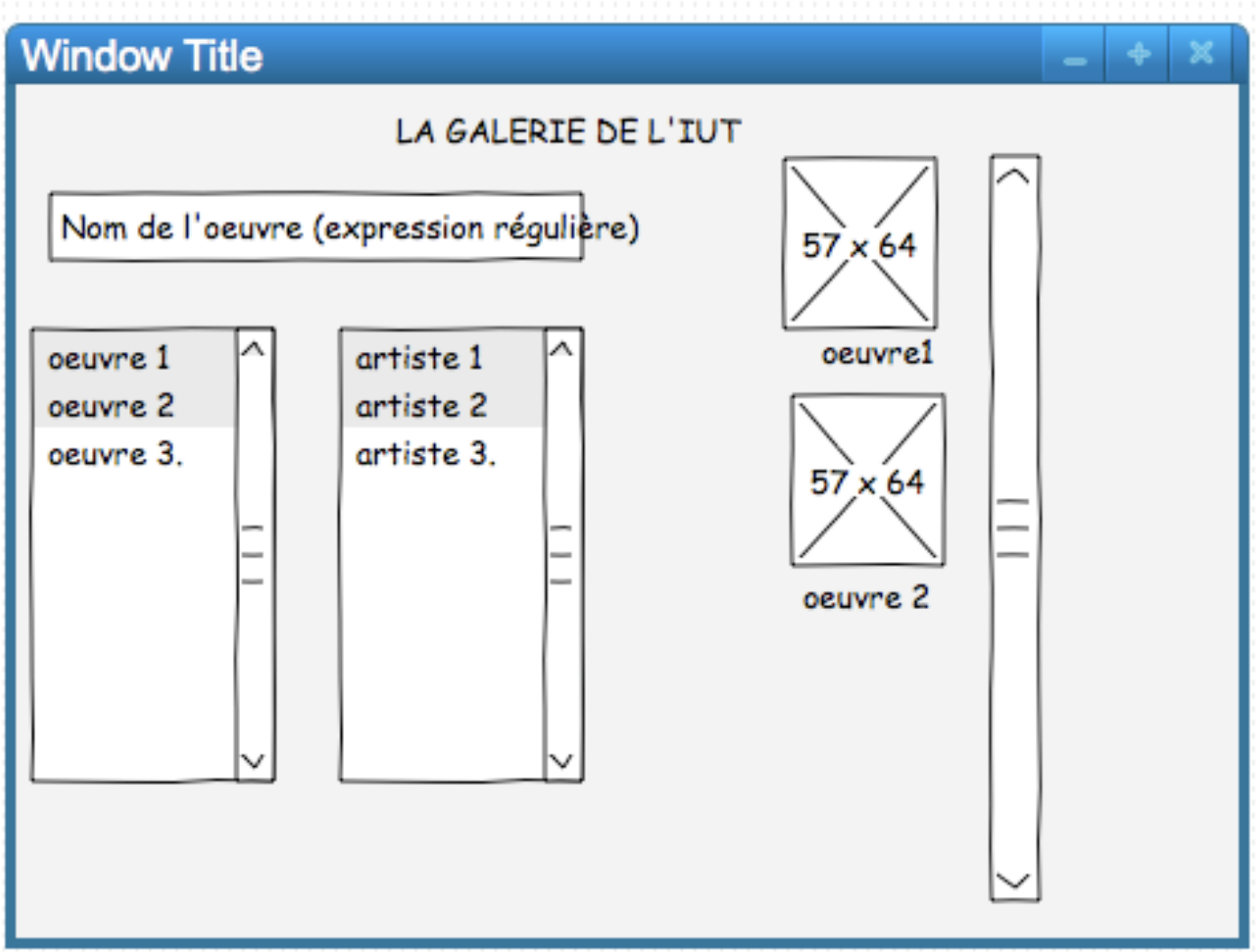
Scenario : Identification des interfaces utilisateur



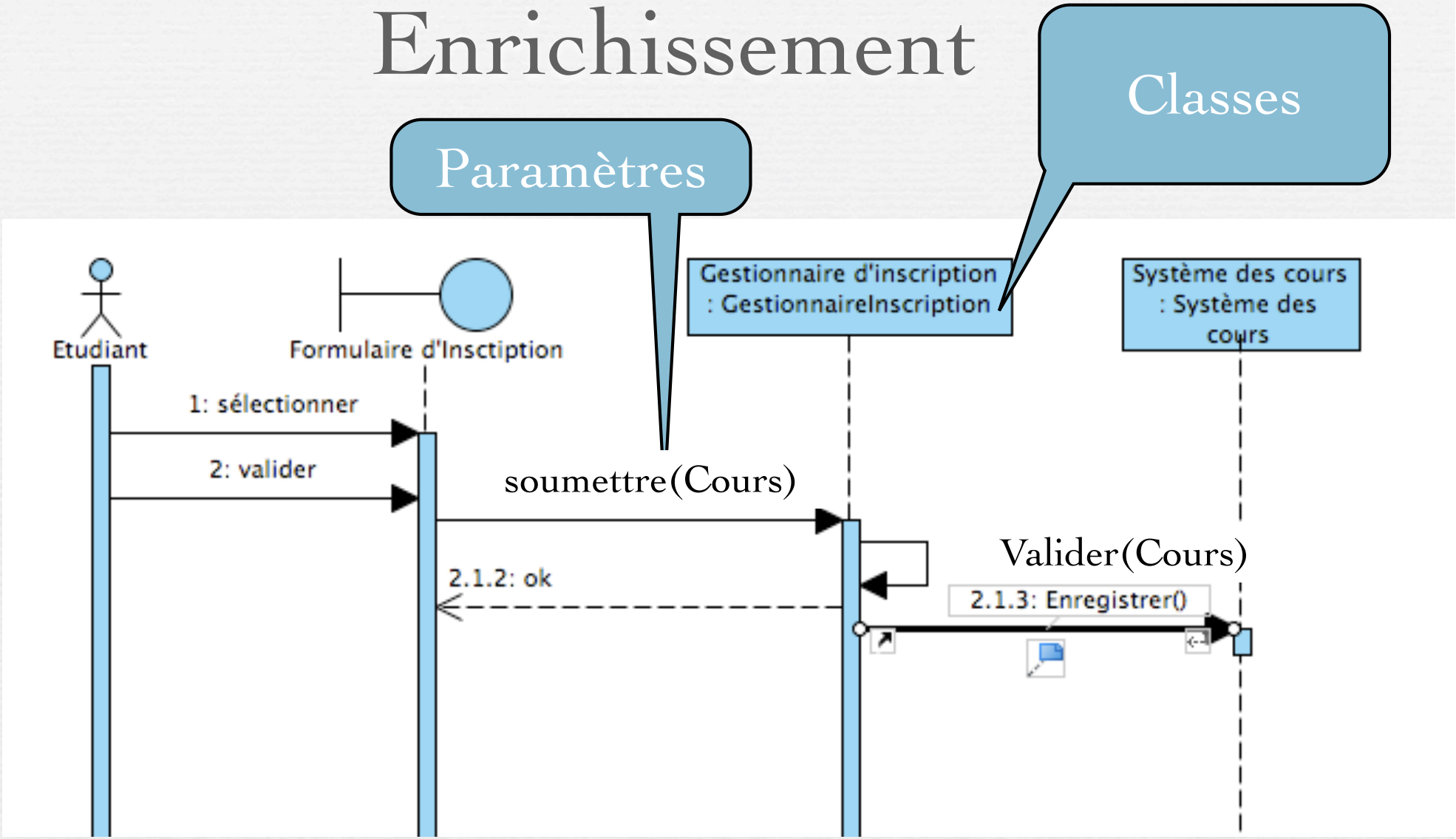
Vue IHM	Description
Sélection Equipe	Sélection d'une équipe dans une liste triée des équipes du laboratoire ou par des critères de recherche.....
Visualisation d'une équipe	Visualisation d'une équipe par ses membres, ...

Si c’est nécessaire...Après...

- avec l’addon Firefox :
- http://pencil.evolus.vn/en-US/Downloads/Application.aspx

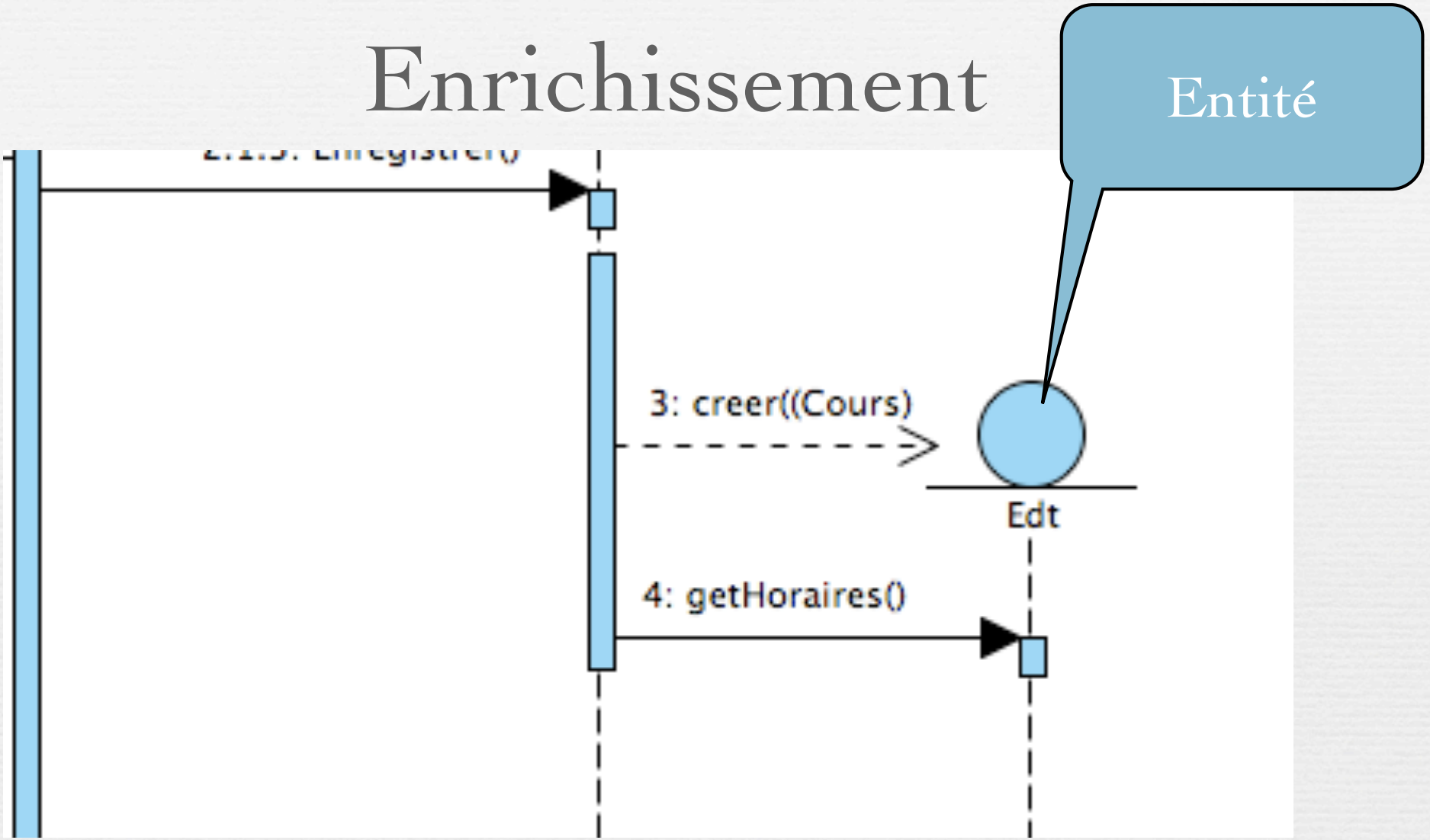


Enrichissement



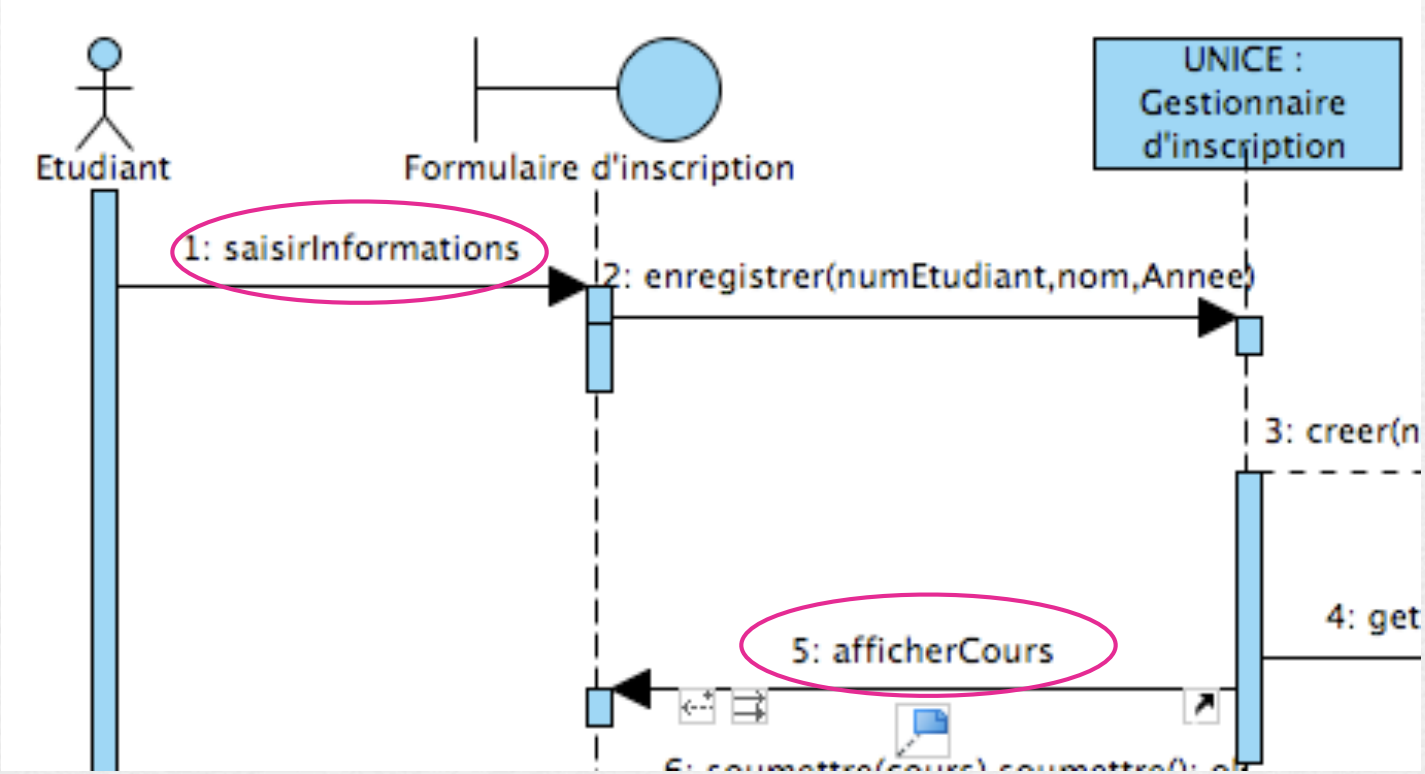
Les “paramètres” sont caractérisés.
Des classes de contrôles apparaissent.

Enrichissement



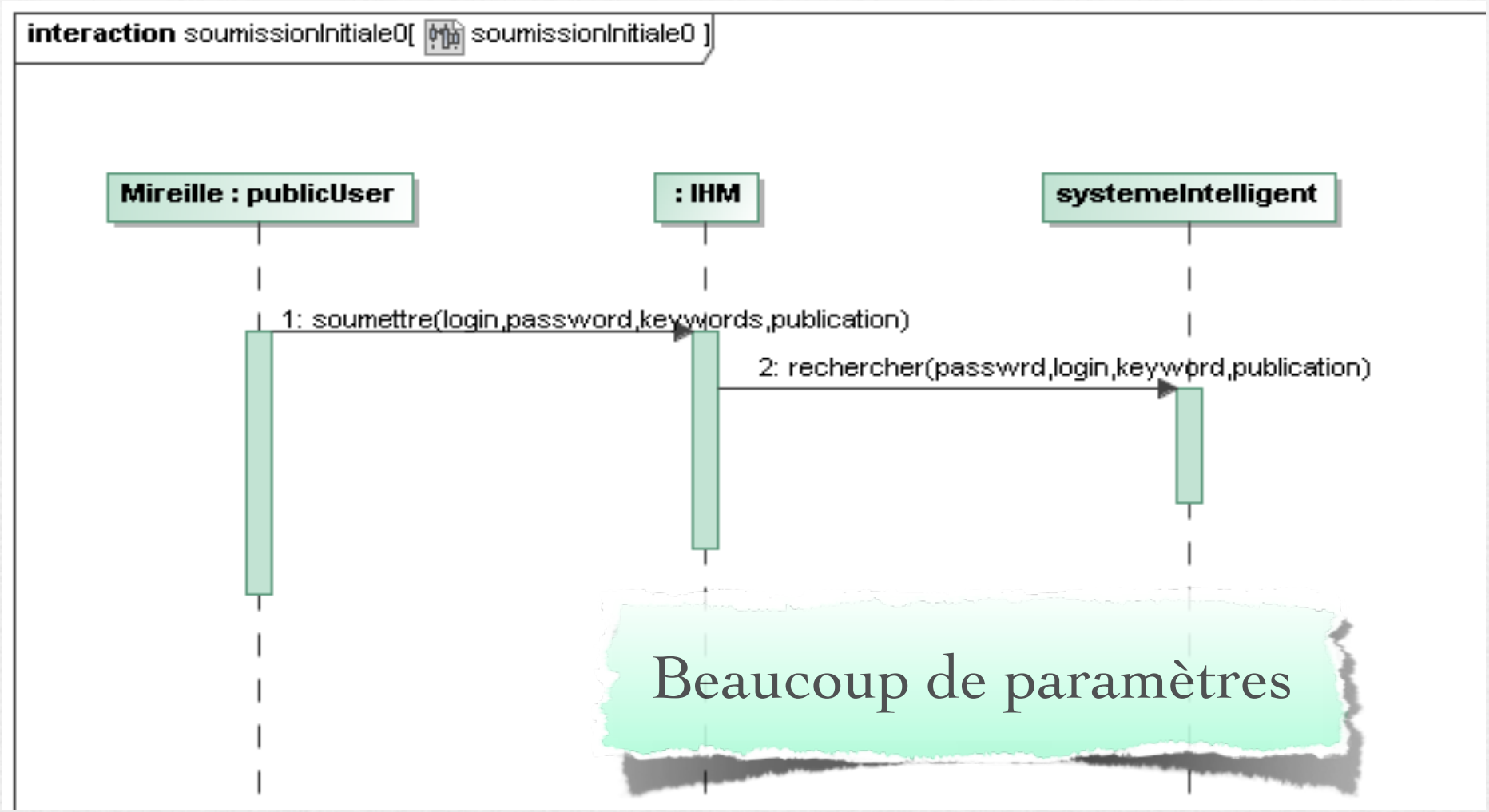
Des données sont identifiées comme persistantes.

Scénario : raffinement

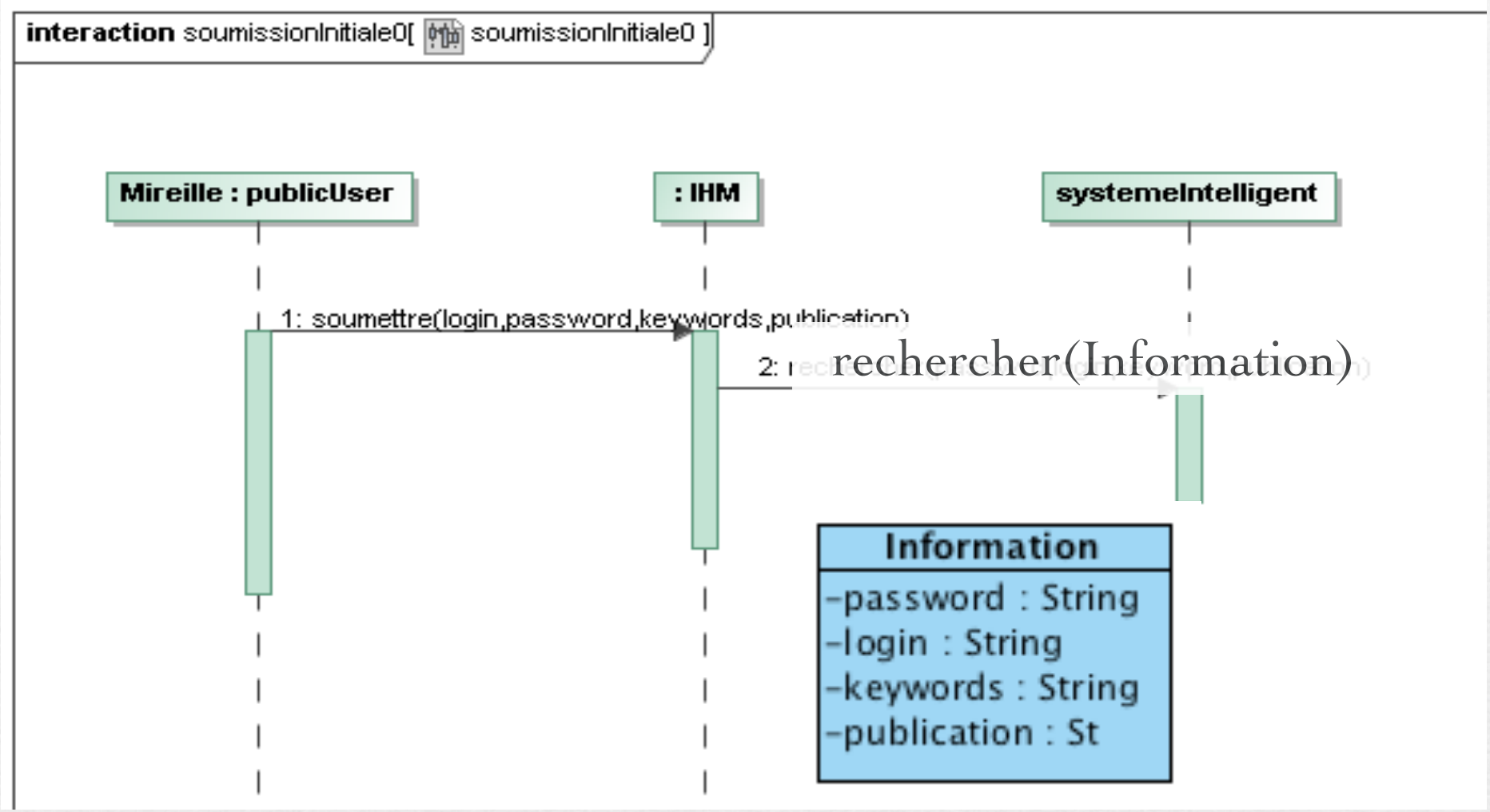


Quelles sont les informations pertinentes ?

Scénario : précisions

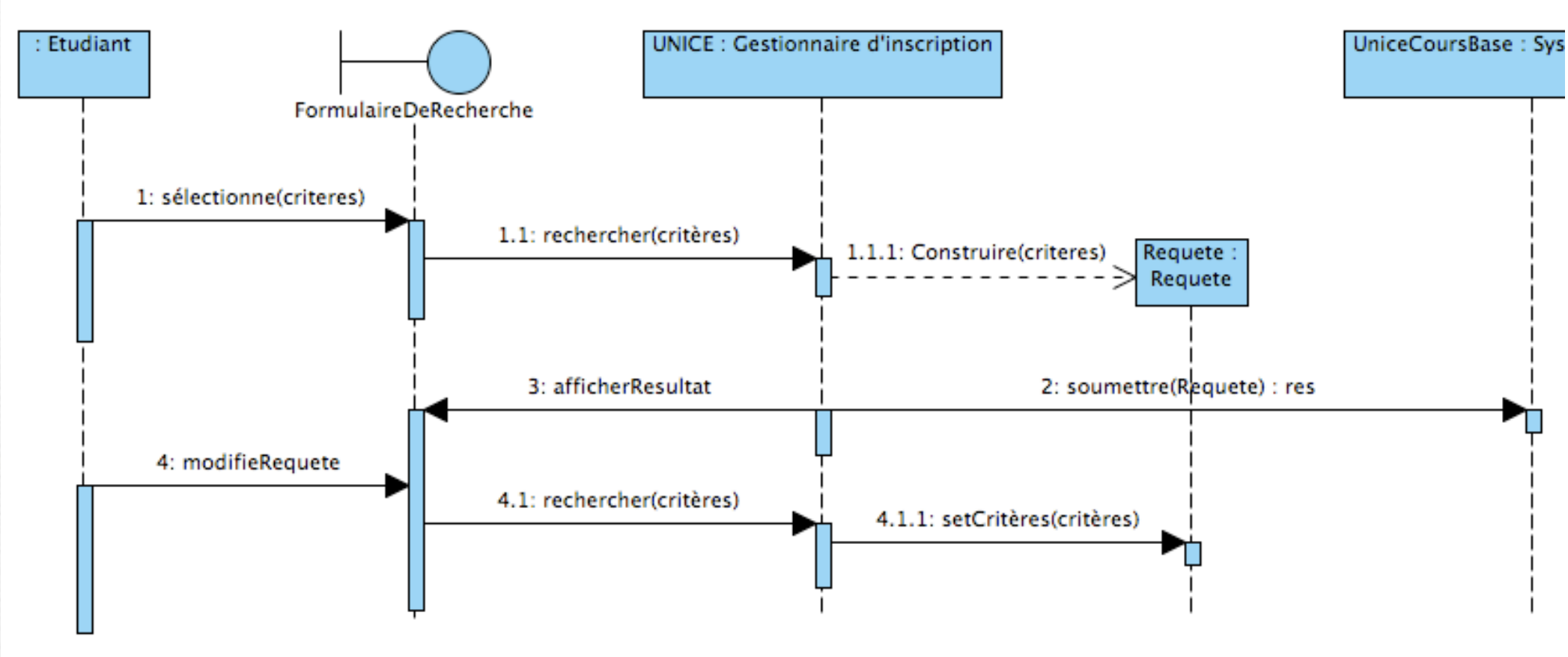


Scénario : précisions



Des “classes” de mise en oeuvre apparaissent

Scenario et réification

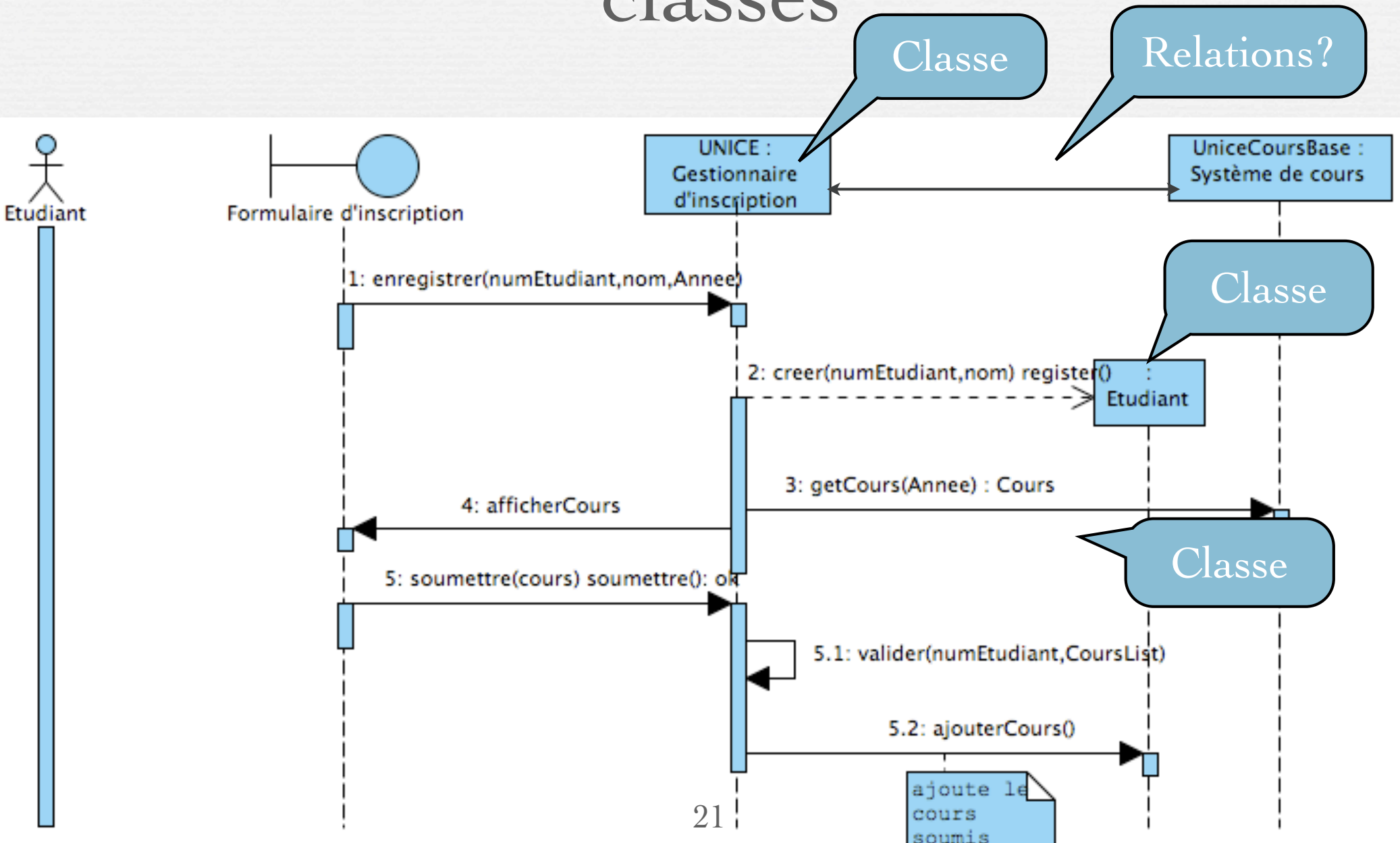


Des “classes” d’implémentation sont identifiées.

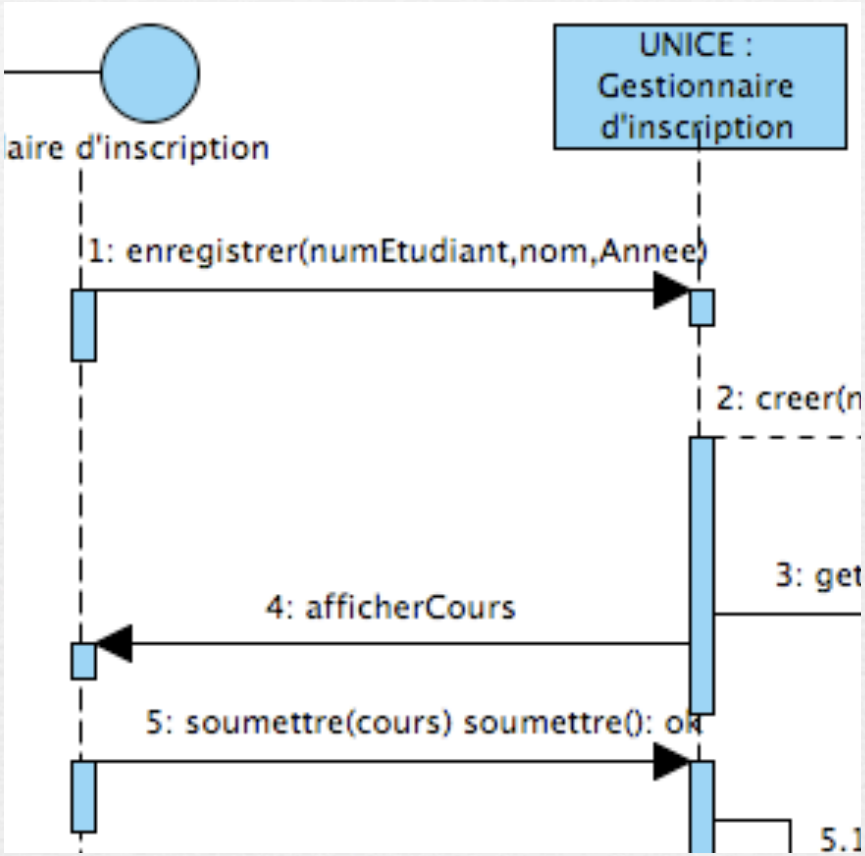
Des diagrammes de séquence et d'activités

aux diagrammes de classes

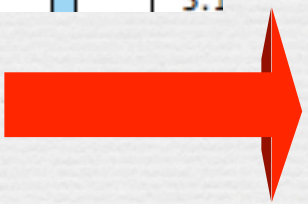
Des diagrammes de séquence aux classes



Opération



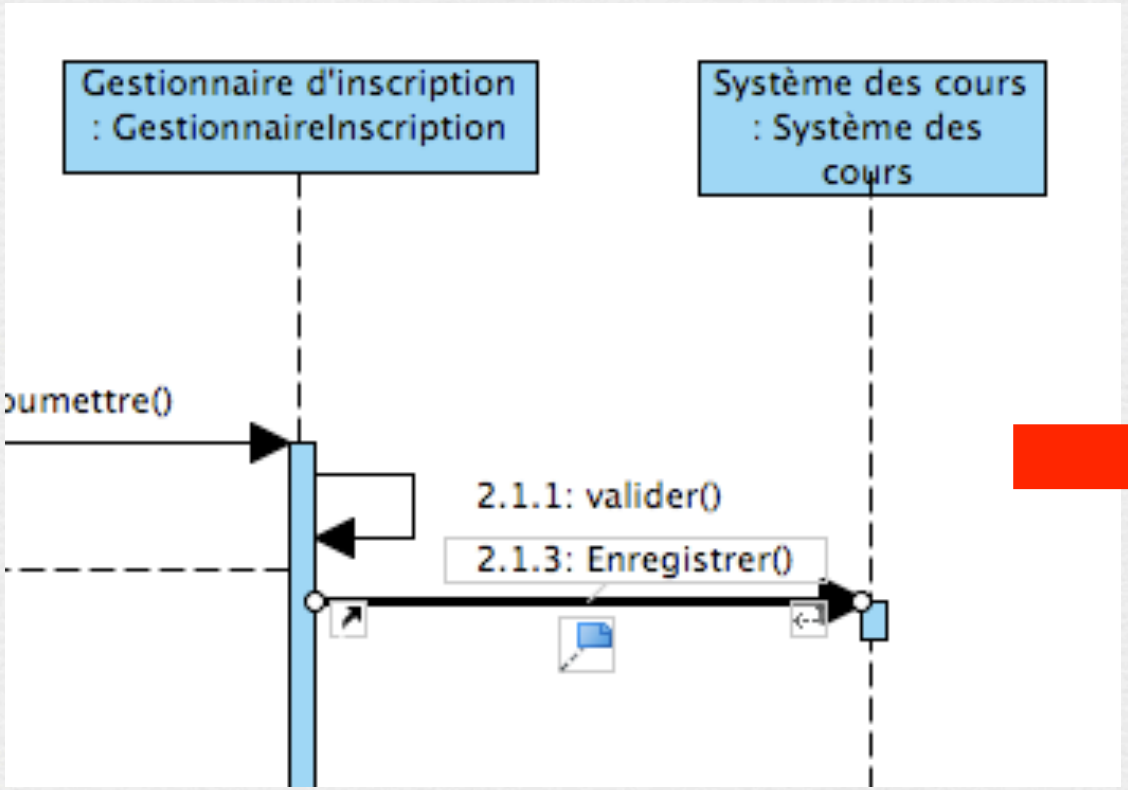
- Le comportement d'une classe est constitué de ses opérations
- On identifie les opérations en examinant les diagrammes d'interactions

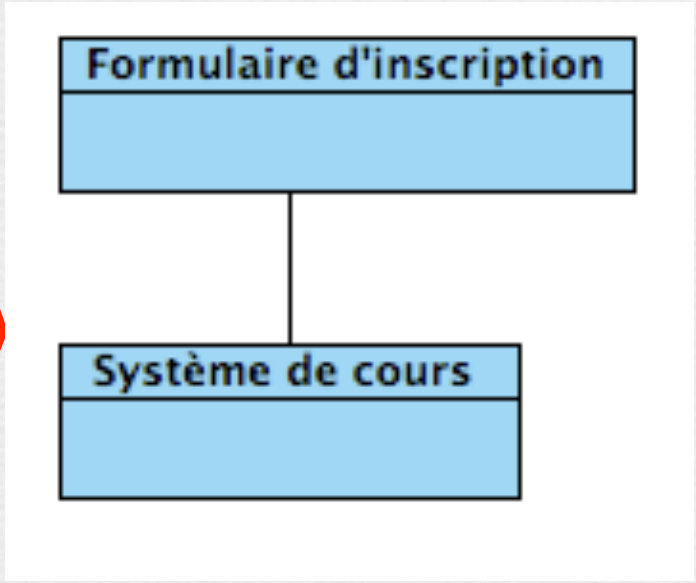


Gestionnaire d'inscription
+soumettre(c : string) : boolean +enregistrer(num : int, nom : string; annee: int)()

Relations

- On identifie les relations en examinant les diagrammes d'interaction
  - Si deux objets doivent communiquer, il doit exister un chemin entre eux





Compléments sur les classes : vers la mise en oeuvre

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

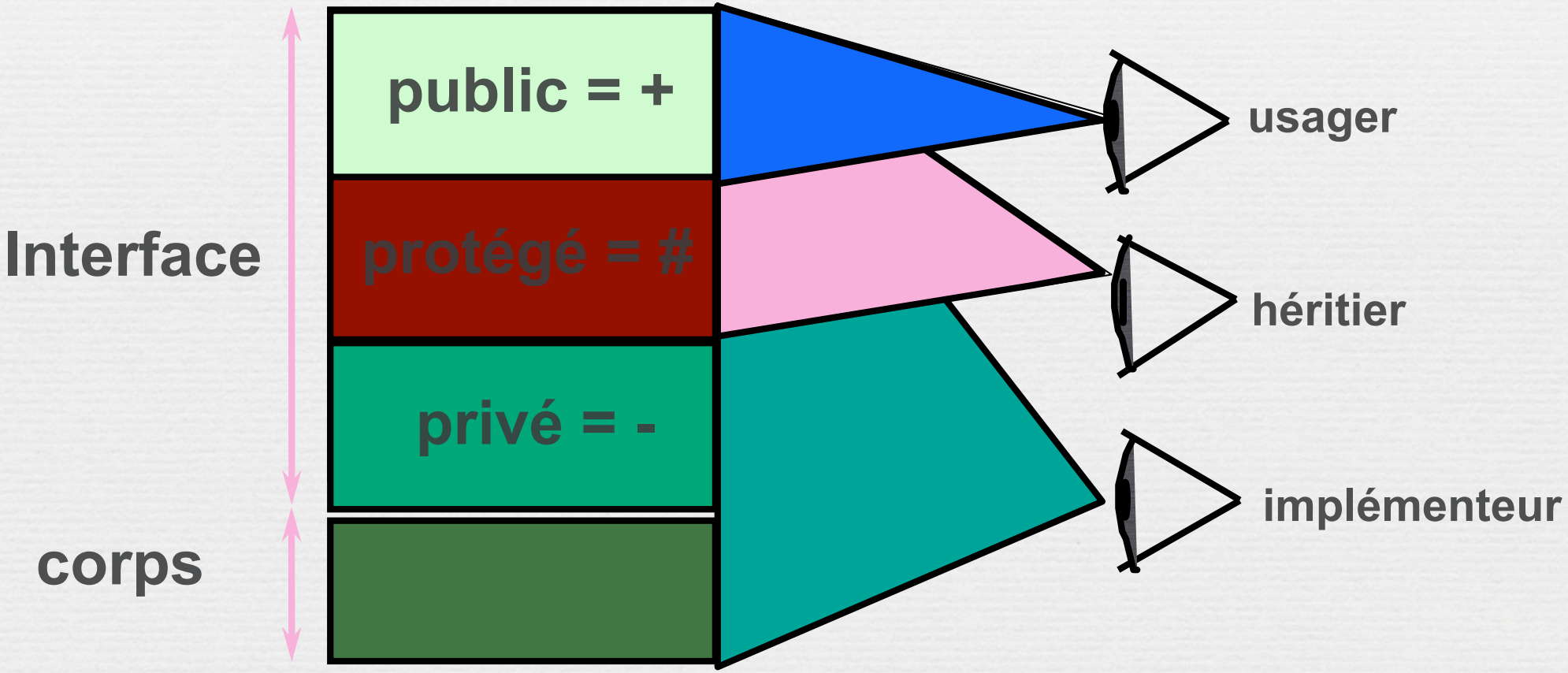
Vers la mise en oeuvre des classes

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

Opération : terme générique désignant le plus souvent des méthodes

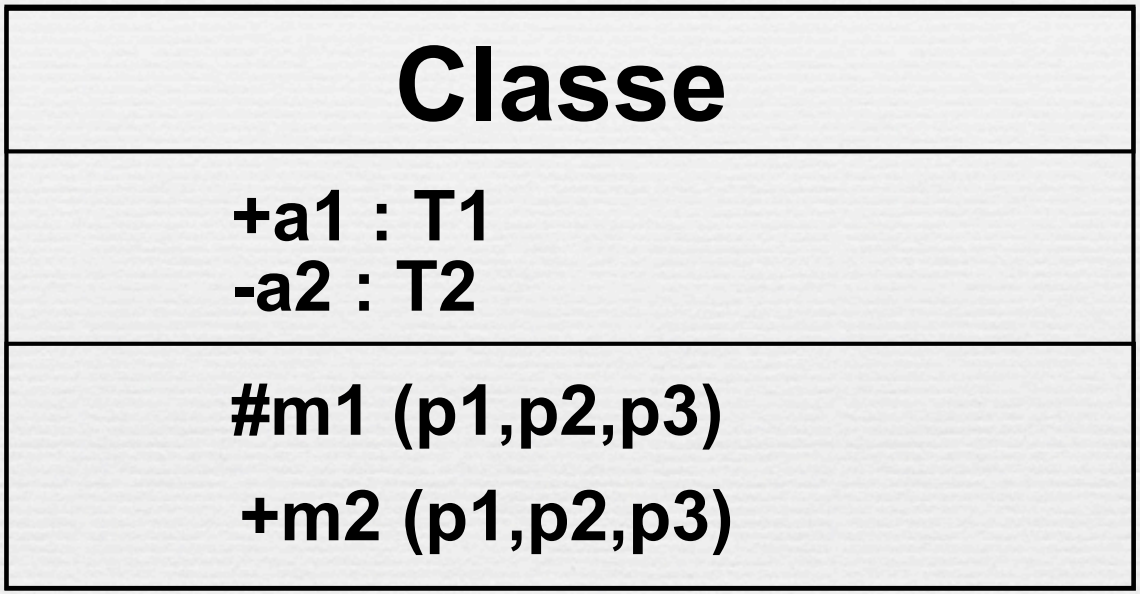
Visibilité

- Différentes visibilités des membres d'une classe



Visibilité

- Représentation



- **Pas de sens en analyse...**

Encapsulation : accessibilité des attributs et des opérations

- Peut-on accéder à tous les attributs ou à toutes les méthodes d'un objet ? Non
  - La classe définit ce qui est accessible
  - C'est le principe de l'encapsulation
  - Un objet complexe ne peut être utilisé qu'au travers de ce qui est accessible
- *Principes :*
  - *Il n'est possible d'utiliser une voiture qu'à travers son volant, son frein, son accélérateur, etc.*
  - *L'accès au carburateur est impossible sauf par les méthodes qui le font de manière cohérente (méthode accélérer de l'accélérateur)*

Encapsulation avec le concept de Visibilité

- Les attributs sont en général inaccessibles (secrets). Ils sont alors qualifiés de :
  - « private » : notation UML « – »
  - Lecture ou modification possible au travers des opérations (p.ex. les ***accesseurs*** : setAdresse(), getAdresse())

- Les opérations sont en général accessibles par toutes les classes. Elles sont alors qualifiées de :
  - « public » : notation UML « + »

Encapsulation avec le concept de Visibilité

- Certains attributs/opérations doivent être accessibles par les sous-classes ou aux classes d'un même package et inaccessibles aux autres classes. Ils sont alors qualifiés de :
  - « protected » : notation UML « # »
- Certaines opérations peuvent cependant
  - être privées (factorisation interne de traitements) et
  - certains attributs peuvent être publics (non souhaitable / principe d'encapsulation)

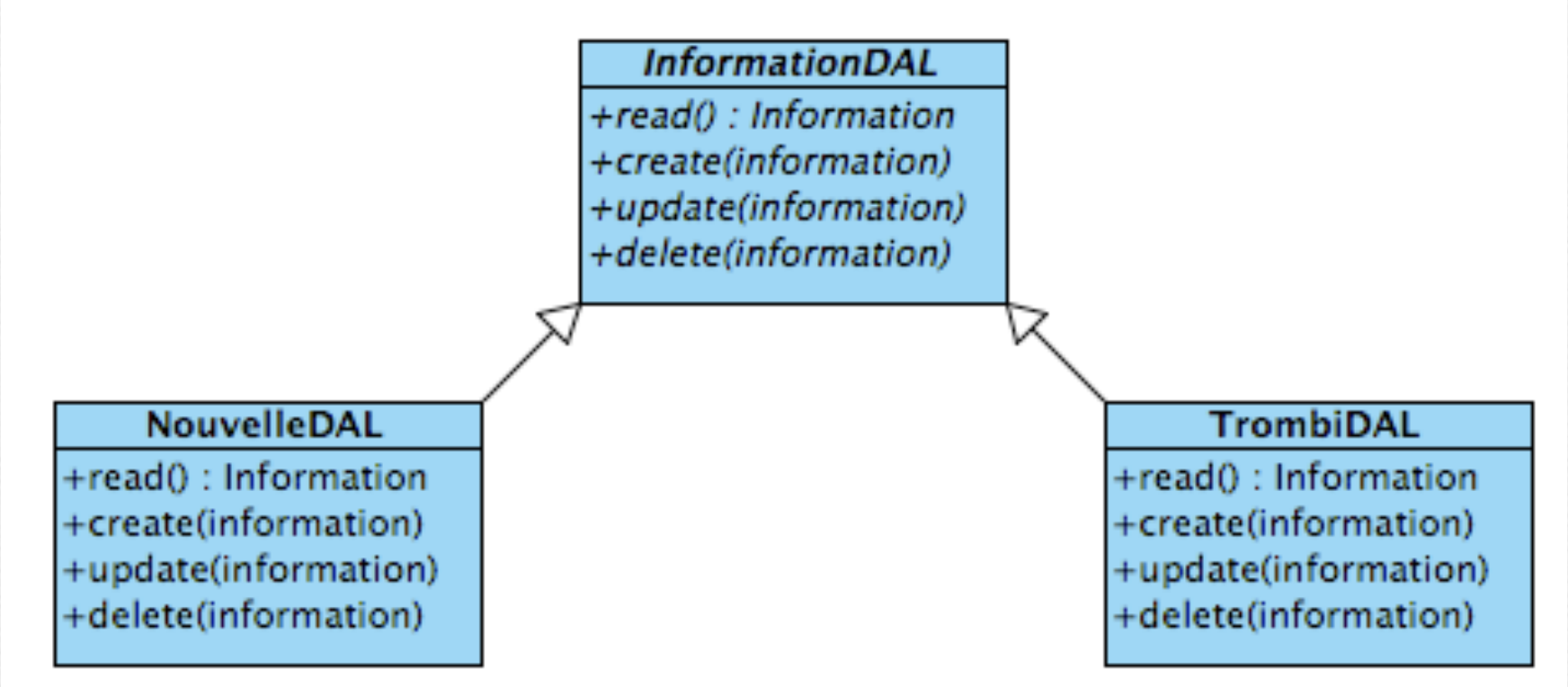
Vers la mise en oeuvre des classes

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

Opération : terme générique désignant le plus souvent des méthodes

Classes et Opérations abstraites

- Une opération/Classe abstraite apparaît en *italique.*

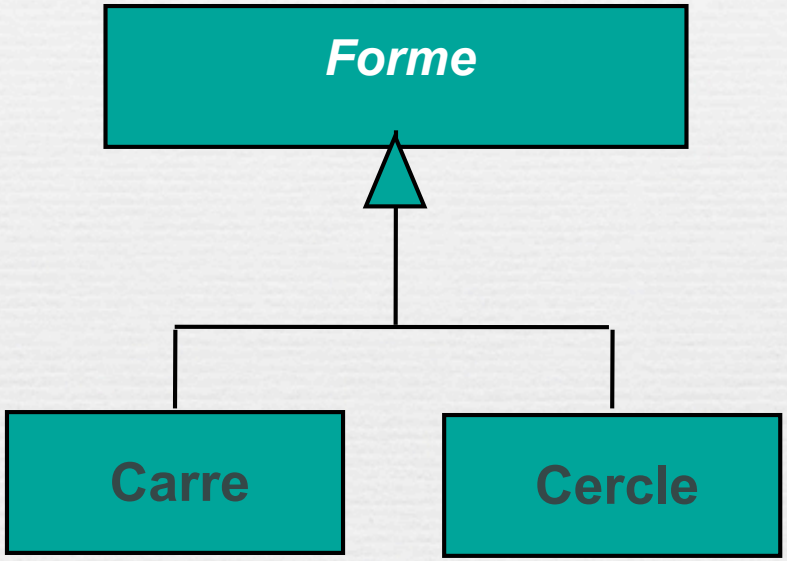


Classes et Opérations abstraites

- Une *classe abstraite* est une classe non instanciable, c'est à dire qu'elle n'admet pas d'instances directes.
- Une classe abstraite est une description d'objets destinée à être « héritée » par des classes plus spécialisées.
- Pour être utile, une classe abstraite doit admettre des classes descendantes *concrètes*.
- La factorisation optimale des propriétés communes à plusieurs classes par généralisation nécessite le plus souvent l'utilisation de classes abstraites.

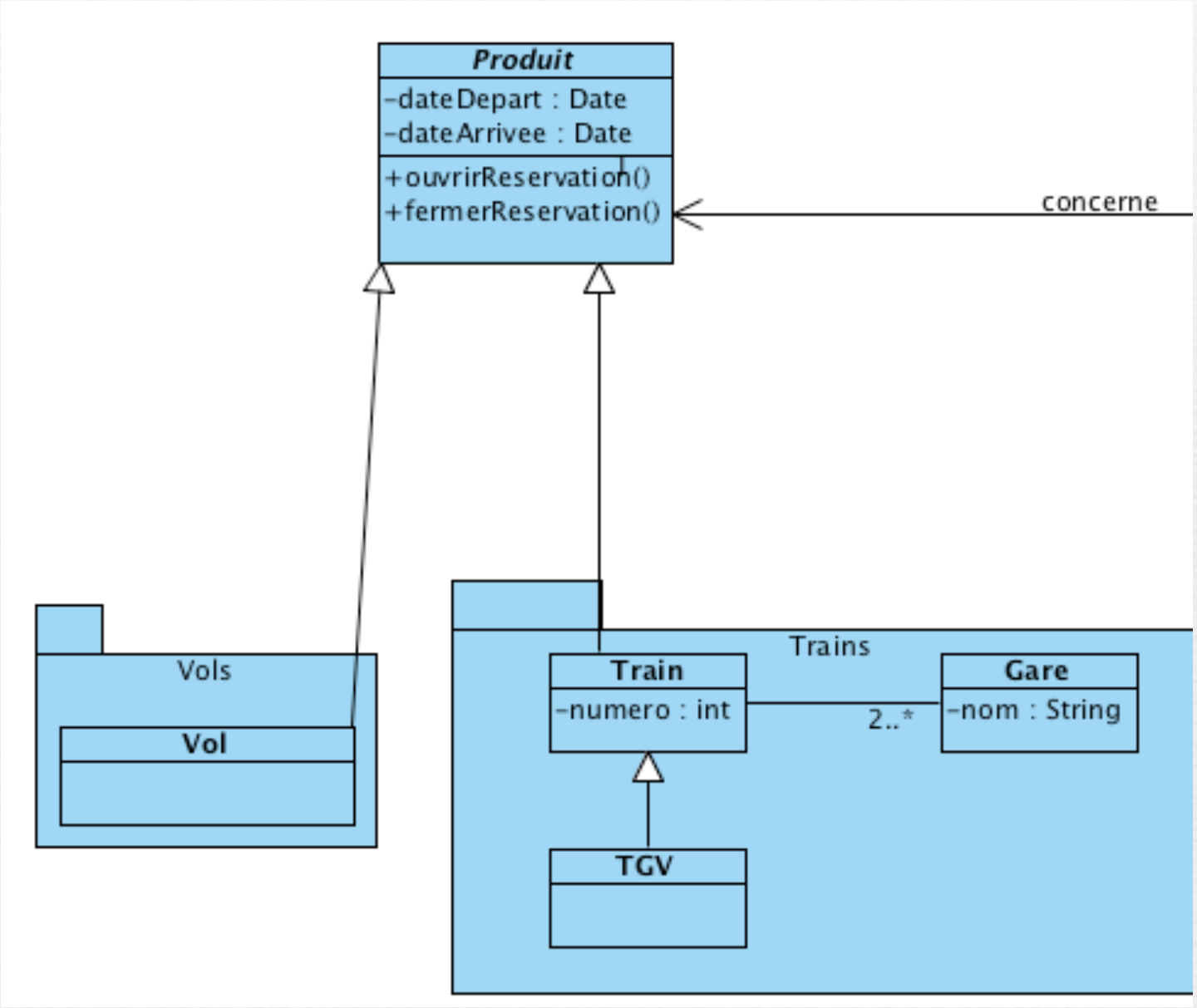
Représentation de classes abstraites

- Classes sans instances immédiates

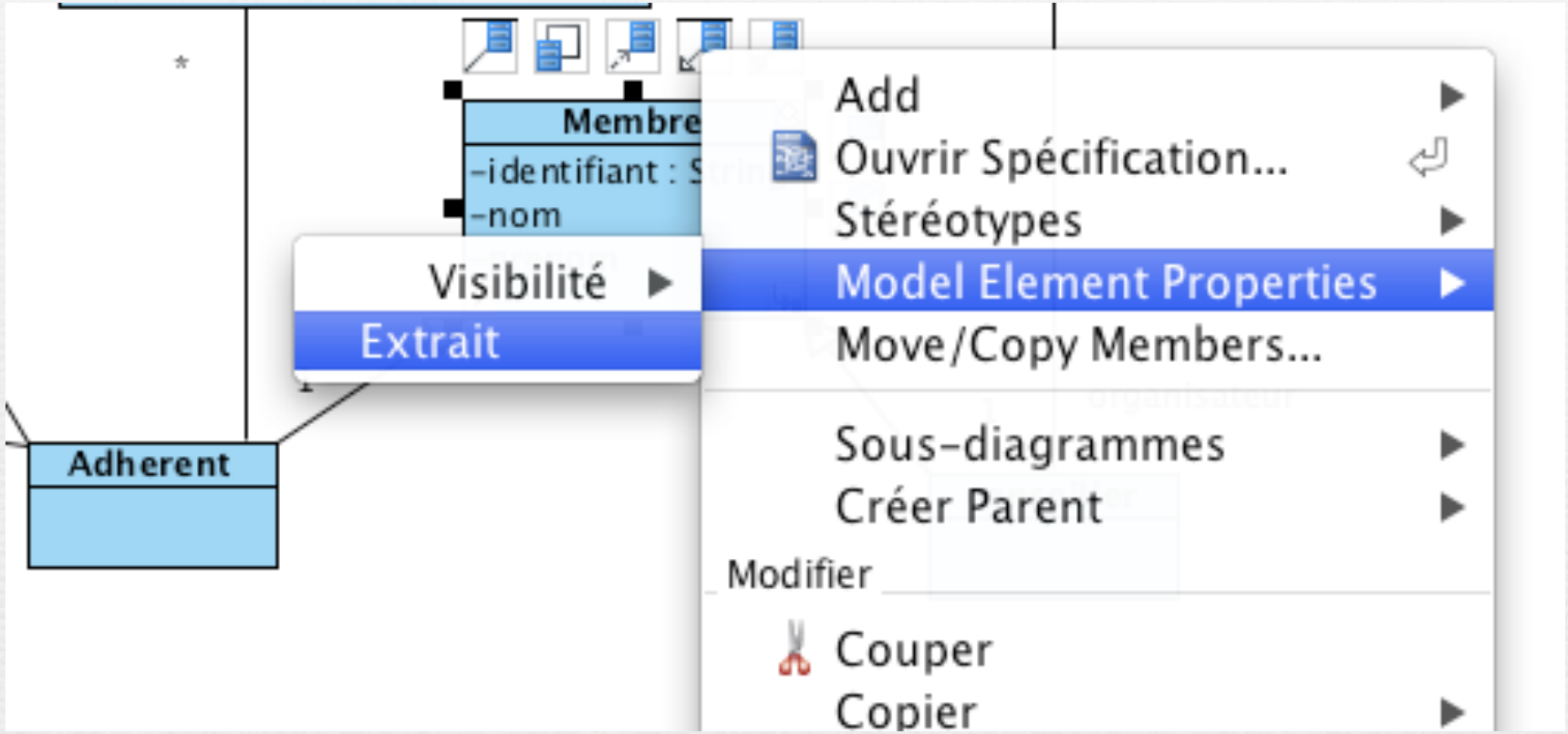


Une instance de «**Forme**» est obligatoirement une instance de la classe **Carre** ou de la classe **Cercle**

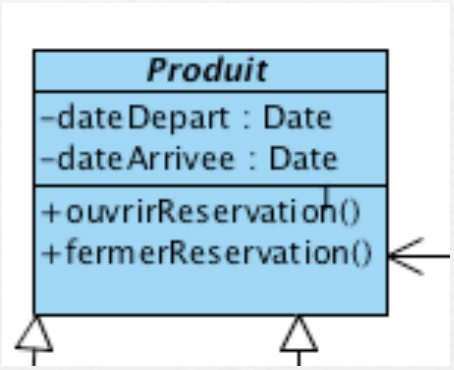
Représentation de classes abstraites



Classes et Opérations abstraites



Représentation de classes abstraites



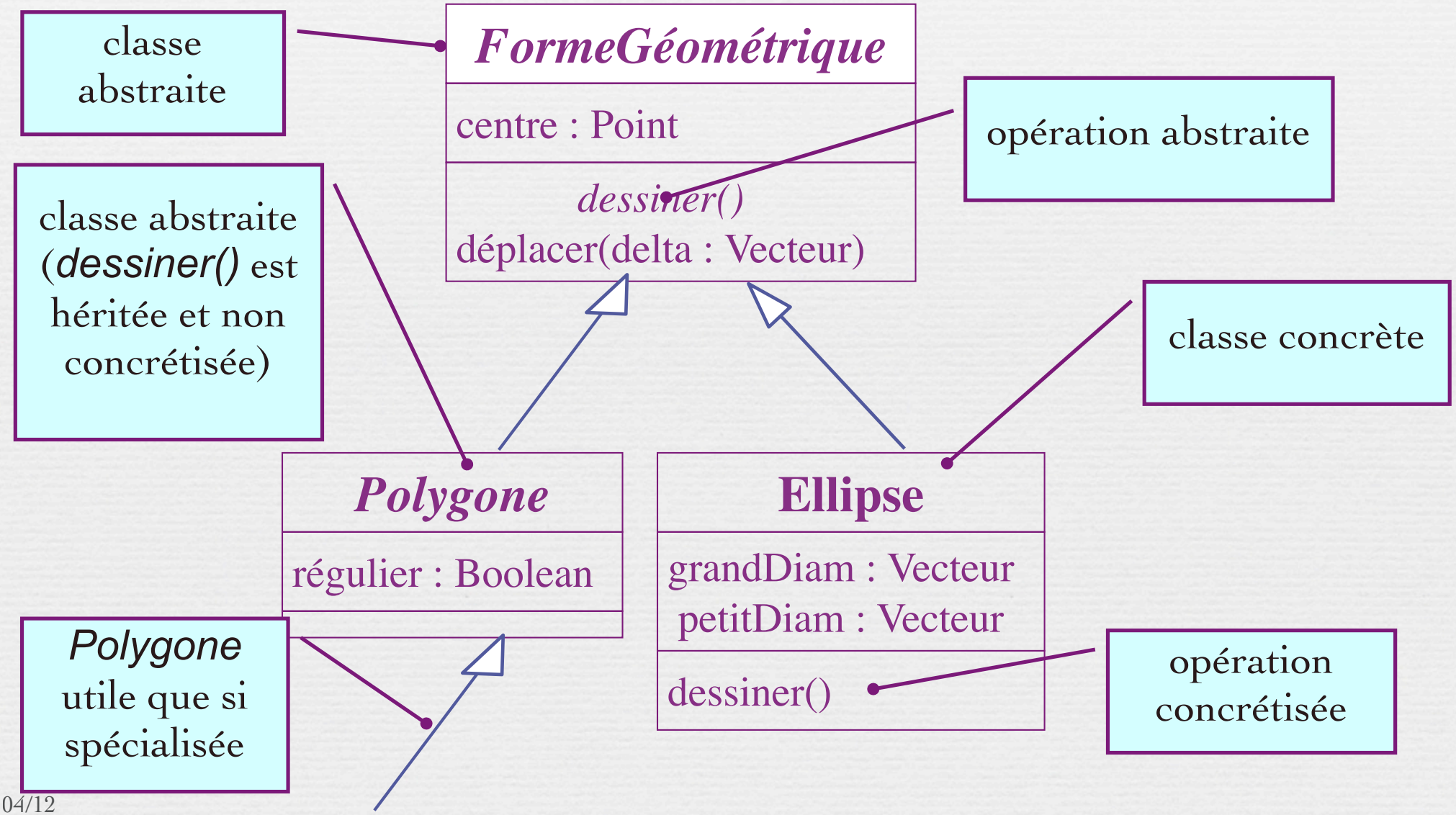
Attention des choix de mises en oeuvre non explicités au niveau du modèle apparaissent dans ce code.

```
package produitPK;

import java.util.Date;

public abstract class Produit {
 private Date dateDepart ;
 private Date dateArrivée ;

 //Choix de mise en oeuvre
 private boolean open = false;

 //Choix de mise en oeuvre
public void setDateDepart(Date dateDepart) {
    this.dateDepart = dateDepart;
}
//Choix de mise en oeuvre
public Date getDateDepart() {
    return dateDepart;
}
//Choix de mise en oeuvre
public void setDateArrivée(Date dateArrivée) {
    this.dateArrivée = dateArrivée;
}
//Choix de mise en oeuvre
public Date getDateArrivée() {
    return dateArrivée;
}
```

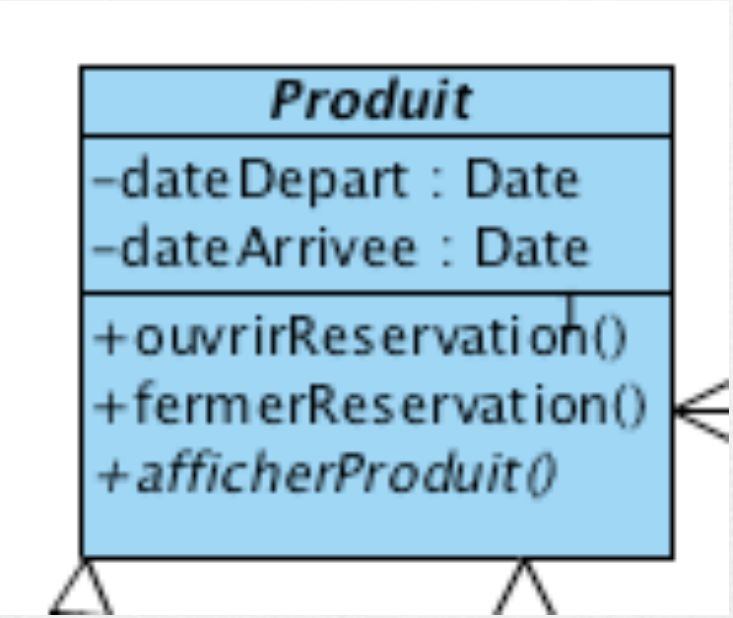
Opérations abstraites

- Une *opération abstraite* est une opération n'admettant pas d'implémentation : au niveau de la classe dans laquelle est déclarée, on ne peut pas dire comment la réaliser.
- Les opérations abstraites sont particulièrement utiles pour mettre en œuvre le polymorphisme.
- Toute classe concrète sous-classe d'une classe abstraite doit “concrétiser” toutes les opérations abstraites de cette dernière.

Classes abstraites



Opérations abstraites



Attention des choix de mises en oeuvre non explicités au niveau du modèle apparaissent dans ce code.

```
public abstract class Produit {
 private Date dateDepart ;
 private Date dateArrivée ;

 //Choix de mise en oeuvre
 private boolean open = false;

 //Choix de mise en oeuvre
public void setDateDepart(Date dateDepart) {
    this.dateDepart = dateDepart;
}
//Choix de mise en oeuvre
public Date getDateDepart() {
    return dateDepart;
}
//Choix de mise en oeuvre
public void setDateArrivée(Date dateArrivée) {
    this.dateArrivée = dateArrivée;
}
//Choix de mise en oeuvre
public Date getDateArrivée() {
    return dateArrivée;
}

public void ouvrirReservation() {
    open = true;
}

public void fermerReservation() {
    open = false;
}

public abstract void afficherProduit();
```

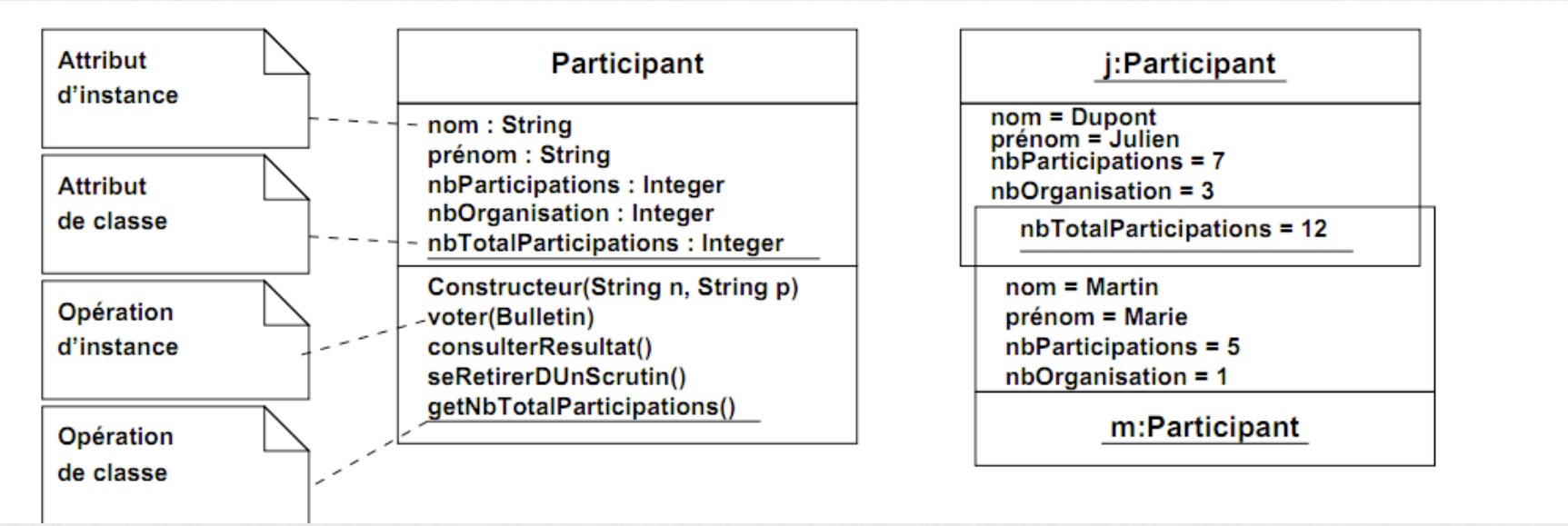
Vers la mise en oeuvre des classes

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

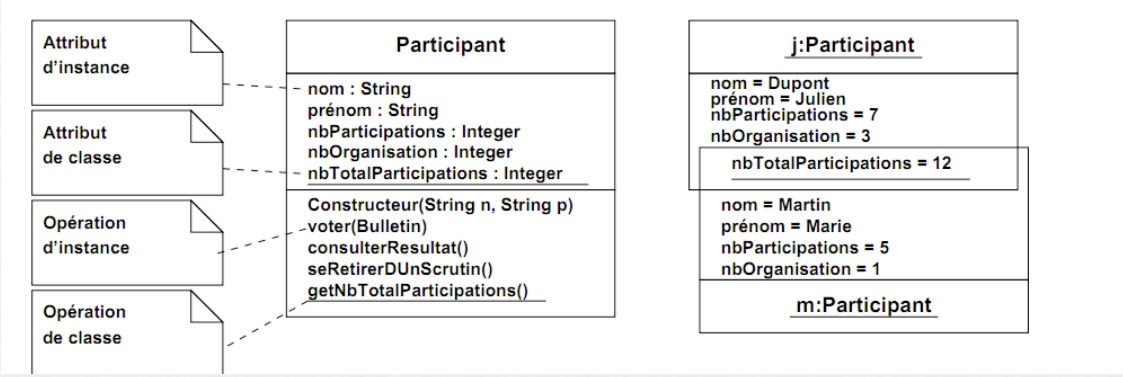
Opération : terme générique désignant le plus souvent des méthodes

Attributs et opérations de classe

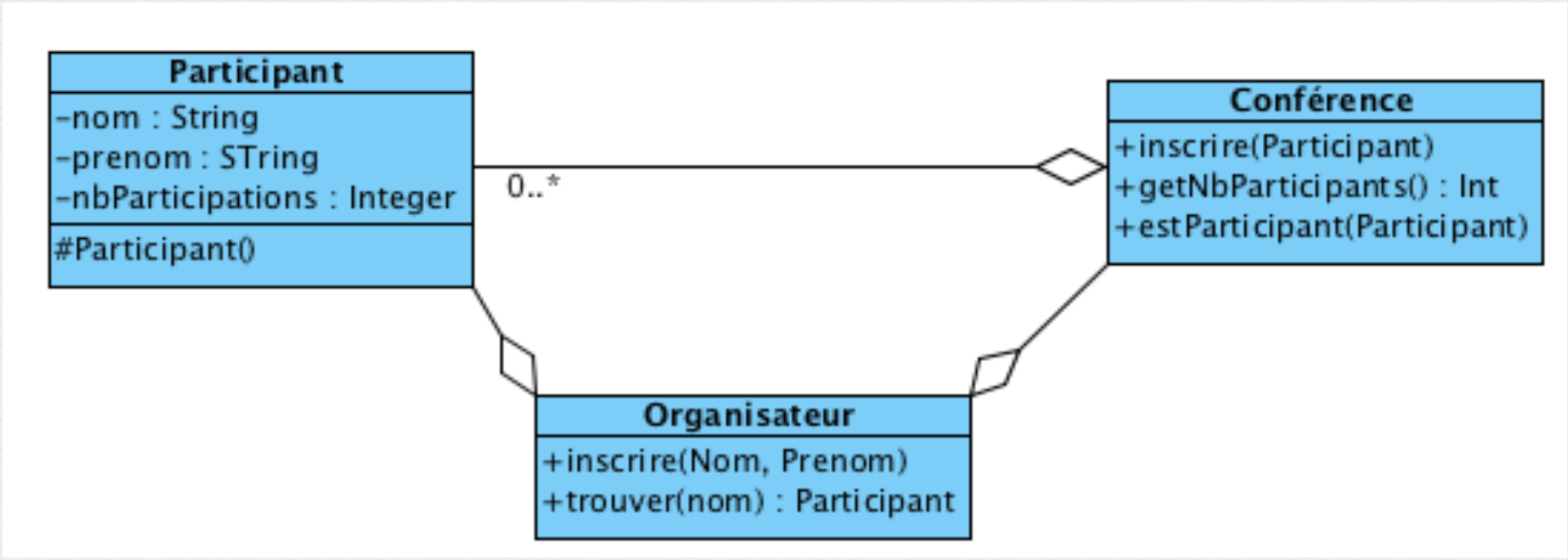
- Le nombre total de participations est une caractéristique des personnes (classe)
- L'opération getNbTotalParticipations() utilise la valeur de l'attribut nbTotalParticipations connue par la classe
  - Cette opération peut être appliquée directement à la classe Personne et aussi aux objets / instances de Personne



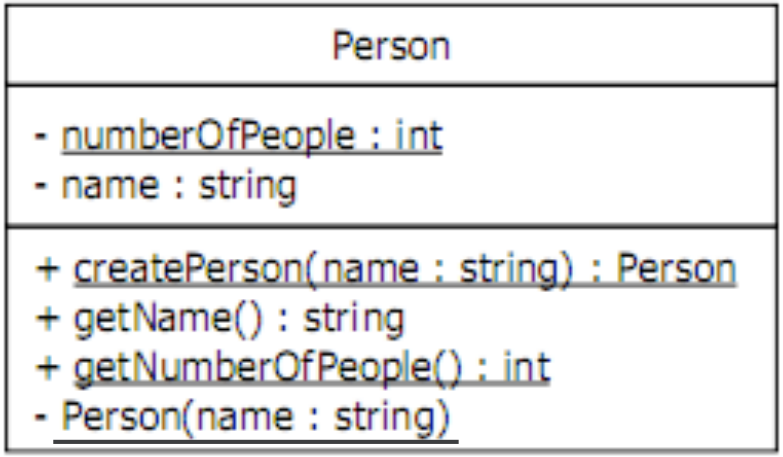
Attributs et opérations de classe versus «fabrique»



A PREFERER EN CONCEPTION.



Attributs et opérations de classe

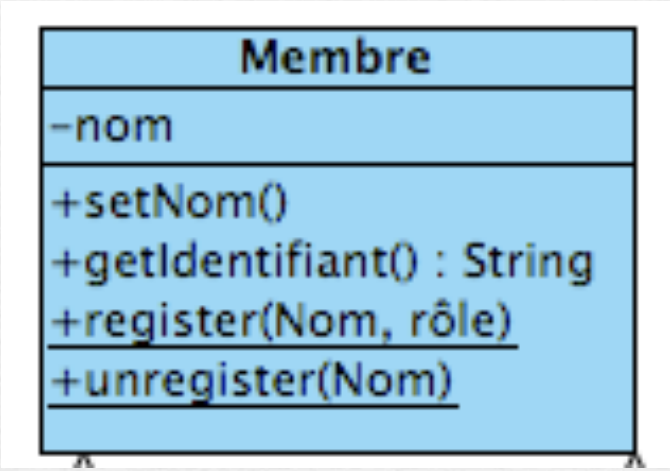


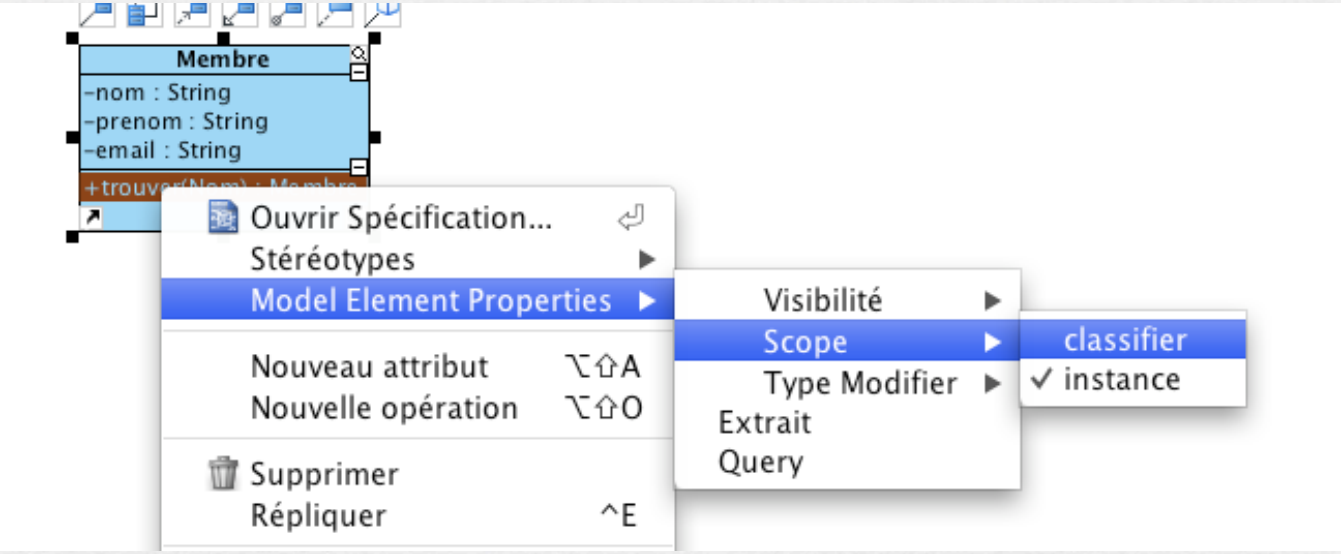

```
int noOfPeople = Person.getNumberOfPeople();
Person p = Person.createPerson("Jason Gorman");
```

```
class Person
{
        private static int numberOfPeople = 0;
        private String name;

        private Person(string name)
        {
                this.name = name;
                numberOfPeople++;
        }

        public static Person createPerson(string name)
        {
                return new Person(name);
        }

        public string getName()
        {
                return this.name;
        }

        public static int getNumberOfPeople()
        {
                return numberOfPeople;
        }
```

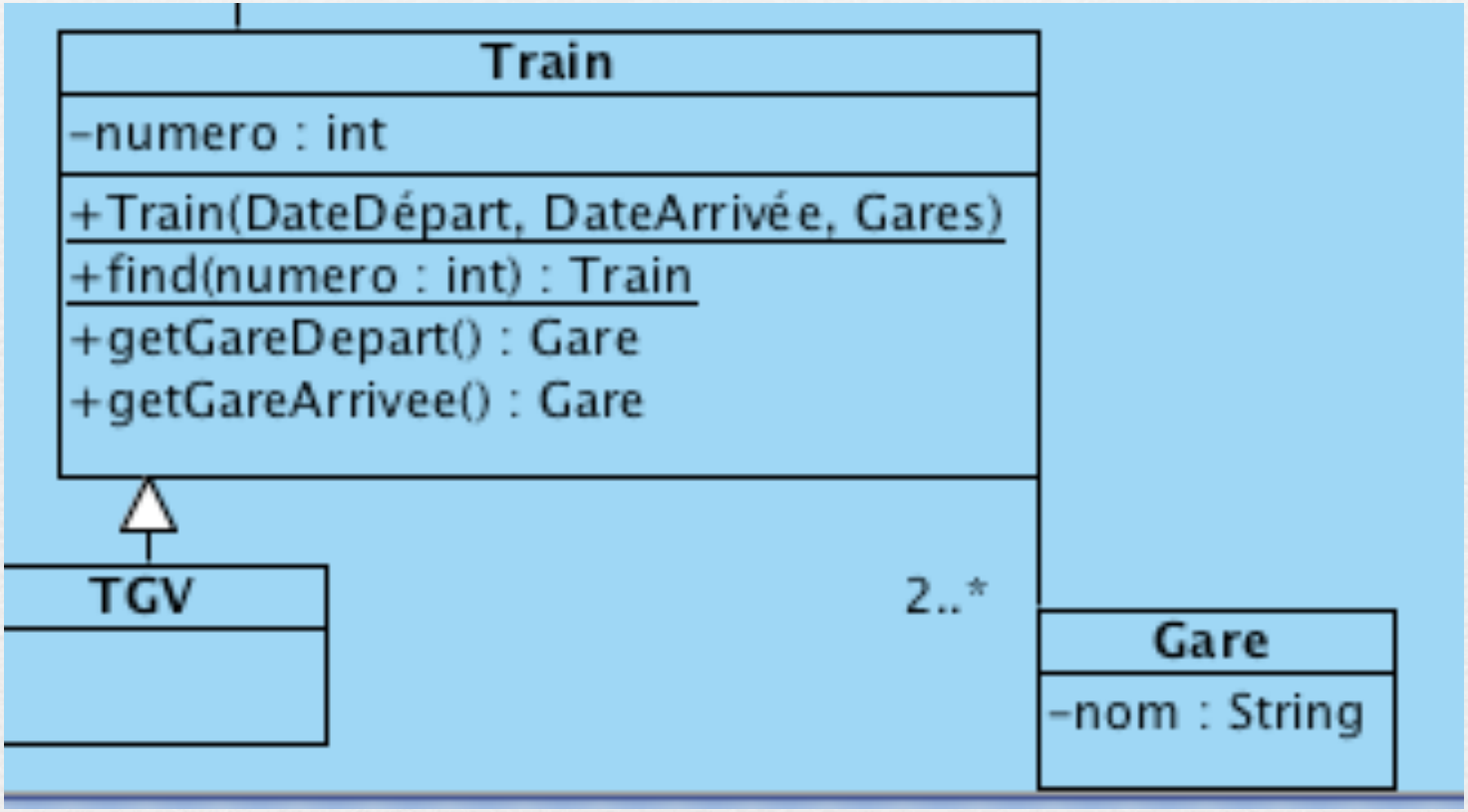
Opérations du niveau de la classe : *Static*

- Une opération «de niveau classe» est soulignée.

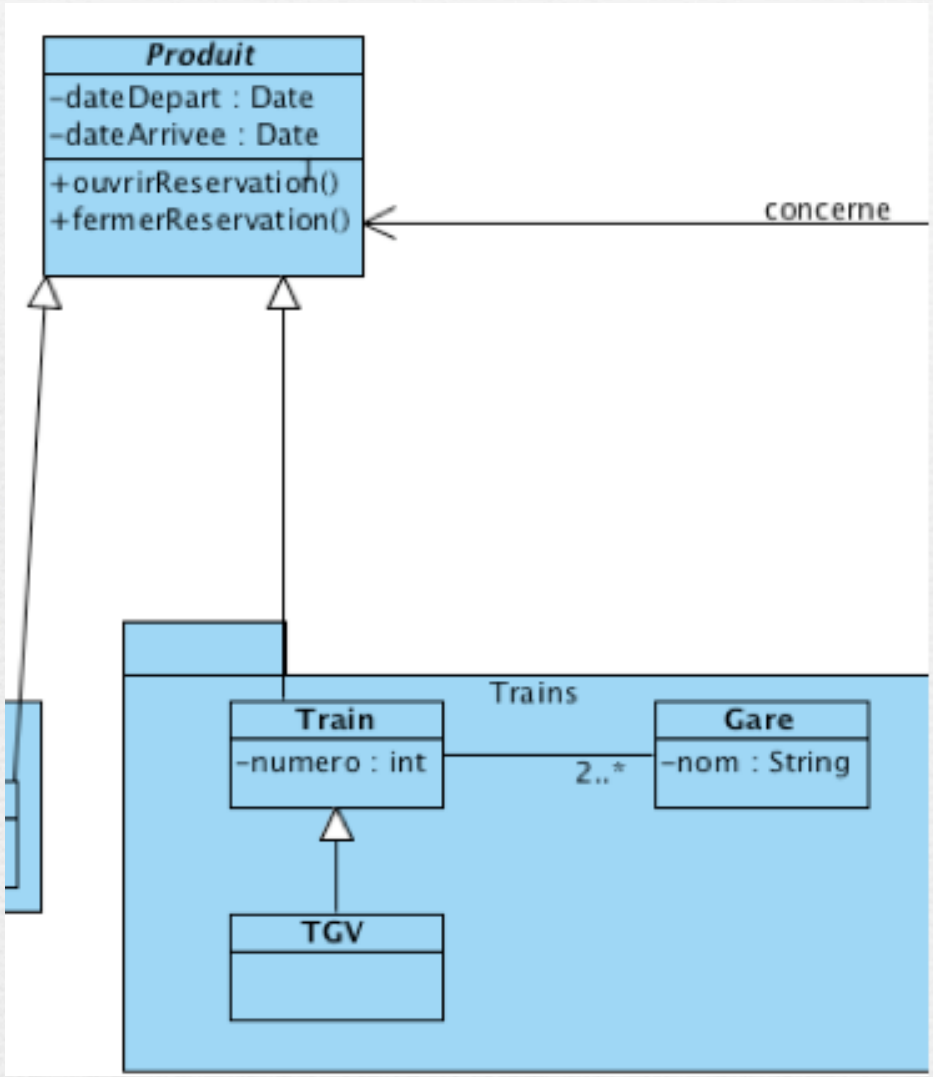




Opérations du niveau de la classe : *Static*



Opérations du niveau de la classe : *Static*



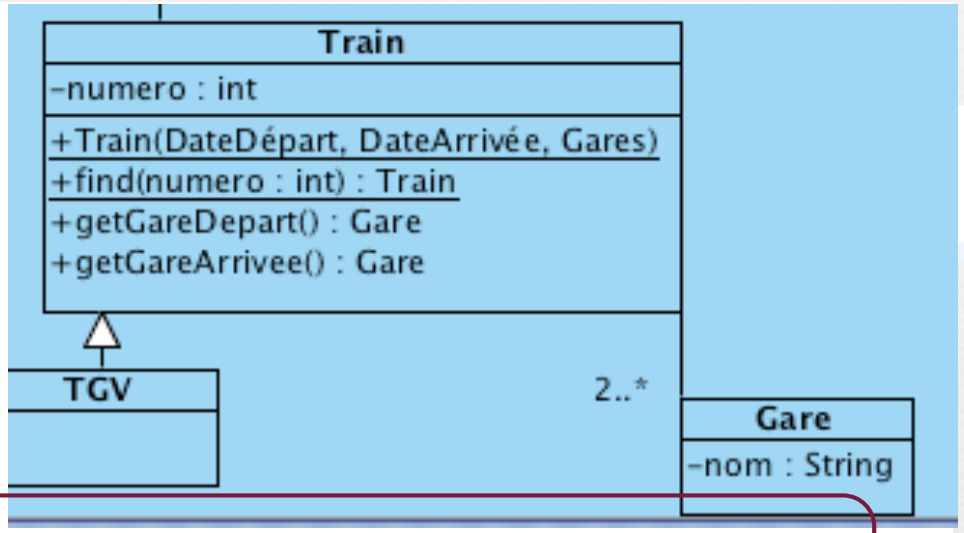
Dans la classe Produit

```
protected Produit(Date dateDepart, Date dateArrivée) {
    this.dateDepart = dateDepart;
    this.dateArrivée = dateArrivée;
}
```

```
package trainPK;

public class Gare {
   String nom;

public String getNom() {
    return nom;
}

public void setNom(String name) {
    this.nom = name;
}

}
```

Opérations du niveau de la classe : *Static*

```
package trainPK;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;

import produitPK.Produit;

public class Train extends Produit{
 int numero;
 Gare[] parcours;

 static private int NombreTrains = 0;
 static private Hashtable<Integer, Train> ListeDesTrains = new Hashtable<Integer, Train>();

 //Constructeur
 public Train(Date DateDepart, Date DateArrivee, Gare[] parcours){

 //Obligatoire
public void afficherProduit() {

public Gare getGareDepart(){
public Gare getGareArrivee(){
public static Train FIND(int numero){

}
```



Opérations du niveau de la classe : *Static*

```
package trainPK;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;

import produitPK.Produit;

public class Train extends Produit{
 int numero;
 Gare[] parcours;

  static private int NombreTrains = 0;
  static private Hashtable<Integer, Train> ListeDesTrains = new Hashtable<Integer, Train>()


  //Constructeur
  public Train(Date DateDepart, Date DateArrivee, Gare[] parcours){
      super(DateDepart,DateArrivee);
      this.parcours = parcours;
      NombreTrains++;
      numero = NombreTrains;
      ListeDesTrains.put(numero,this);
  }
}
```

Opérations du niveau de la classe : *Static*

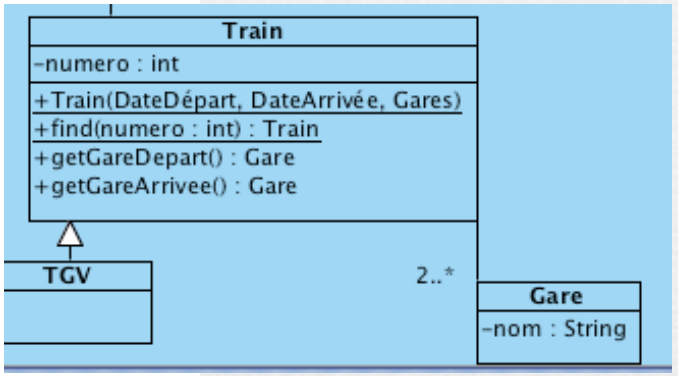
```
package trainPK;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;

import produitPK.Produit;

public class Train extends Produit{
 int numero;
 Gare[] parcours;

  static private int NombreTrains = 0;
  static private Hashtable<Integer, Train> ListeDesTrains = new Hashtable<Integer, Train>()


  public static Train FIND(int numero){
      return ListeDesTrains.get(numero);
  }
    ListeDesTrains.put(numero,this);
 }
}
```

Opérations du niveau de la classe : *Static*

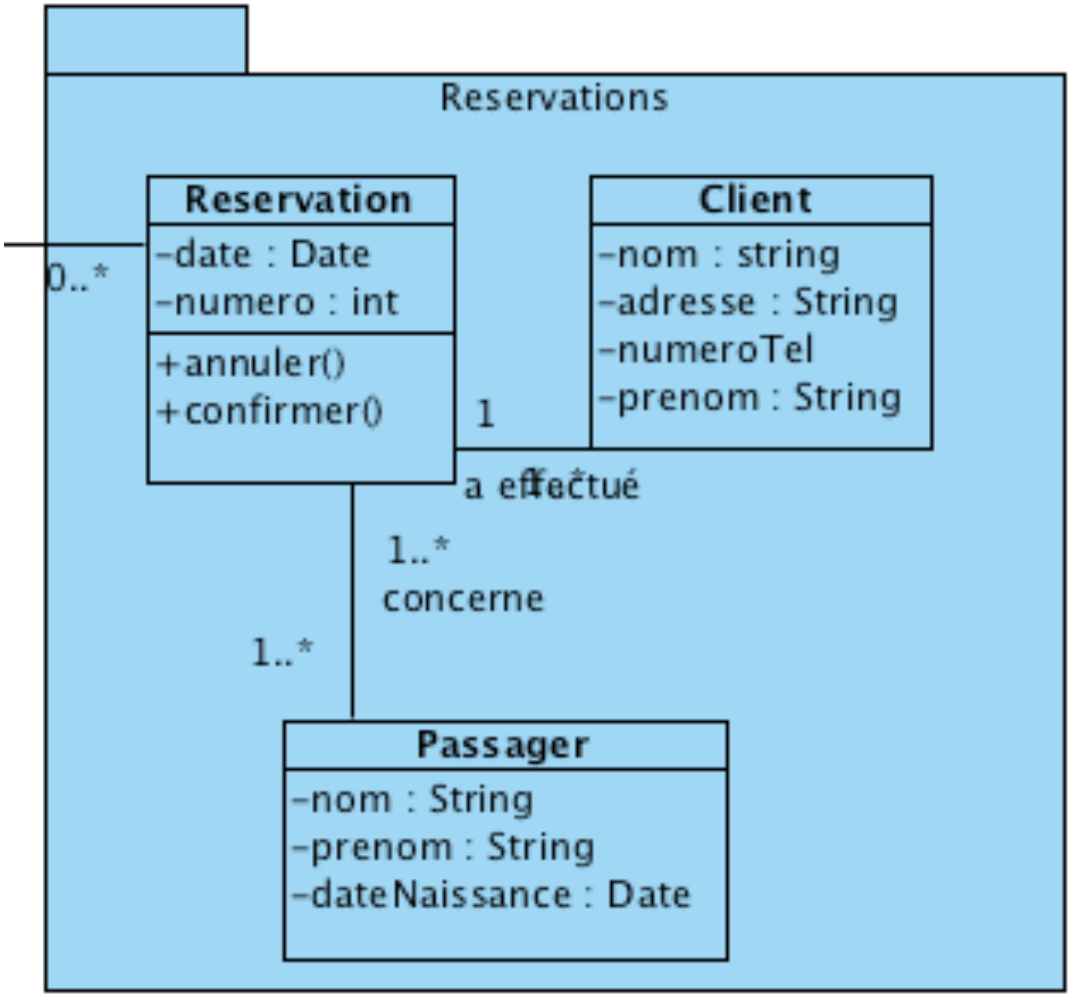
```
package trainPK;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Hashtable;

import produitPK.Produit;

public class Train extends Produit{
 int numero;
 Gare[] parcours;
  //Obligatoire
@Override
 public void afficherProduit() {
     SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd:hh:mm");
     System.out.print("train " + numero + " de ");
     System.out.print(parcours[0].getNom());
     System.out.println( " a " + parcours[parcours.length-1].getNom());
     System.out.println( dateFormat.format(this.getDateDepart()) + " -- " +
                         dateFormat.format(this.getDateArrivee()) );
 }

public Gare getGareDepart(){
     return parcours[0];
 }
public Gare getGareArrivee(){
     return parcours[parcours.length-1];
```

Opérations du niveau de la classe : *Utilisation*

```
public class TestTrains {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception{
        //Pas joli : il faudrait un constructeur
        Gare nice = new Gare();
        nice.setNom("Nice");
        Gare antibes = new Gare();
        antibes.setNom("antibes");

        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd:hh:mm");

        // date to string
        Date depart = dateFormat.parse("2011-03-12:08:00");
        System.out.println("Depart :  "+dateFormat.format(depart));
        Date arrivee = dateFormat.parse("2011-03-12:08:35");
        Train tMatin = new Train(depart, arrivee, new Gare[]{nice,antibes});
        tMatin.afficherProduit();

        Train tSoir = new Train(dateFormat.parse("2011-03-12:19:00"),
                dateFormat.parse("2011-03-12:19:40"), new Gare[]{antibes,nice});
        tSoir.afficherProduit();

        System.out.println("---------------- ");
        System.out.println("Train du matin ");
        Train.FIND(1).afficherProduit();
        System.out.println("Train du soir ") ;
        Train.FIND(2).afficherProduit();
    }
```

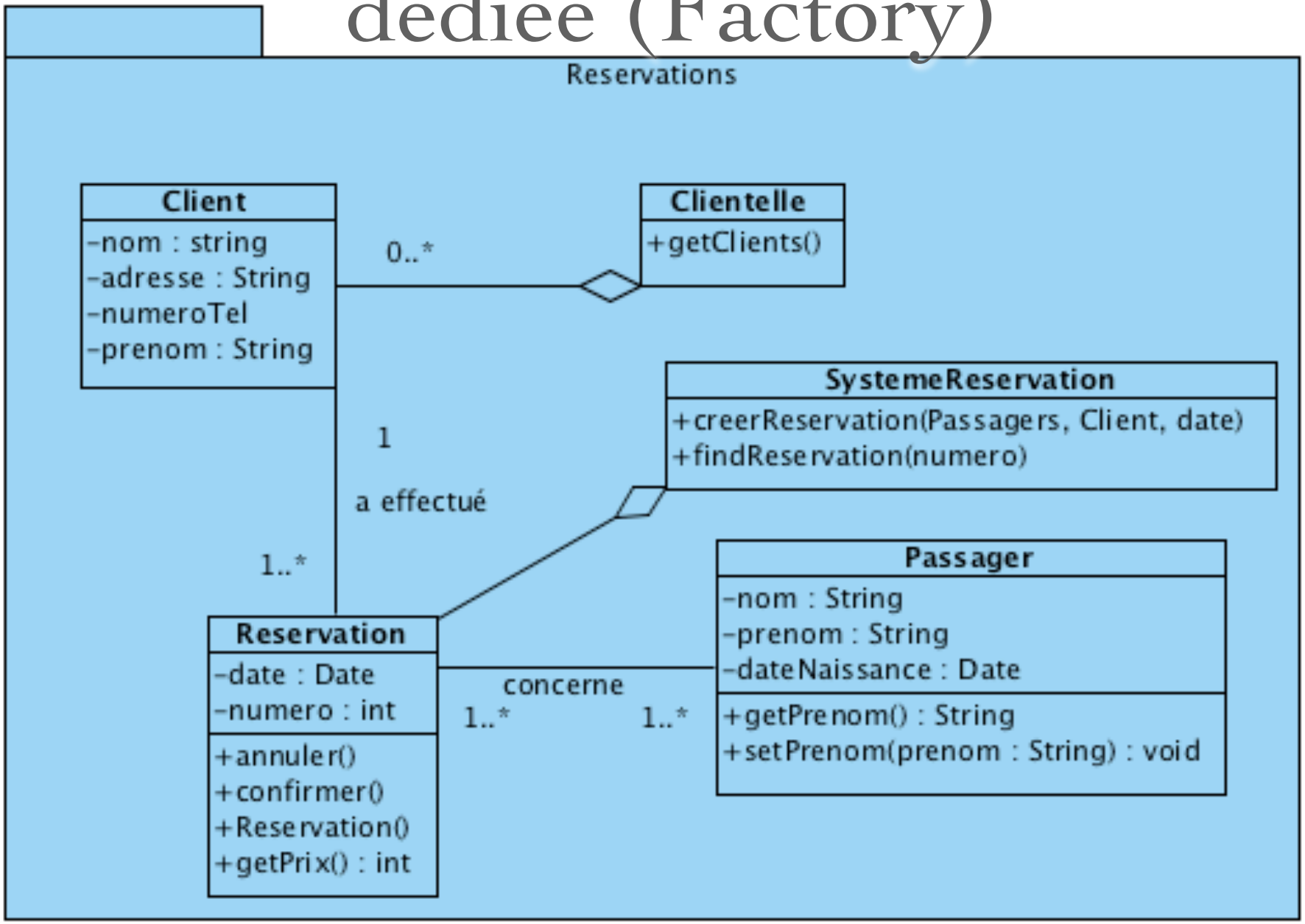


Opérations du niveau de la classe : *Static*



1. Obtenir la liste des clients
2. Modifier la date d'une réservation
3. Créer une réservation
4. Modifier le prénom d'une personne
5. Calculer le prix d'une réservation

Propriété Statique ou Classe dédiée (Factory)

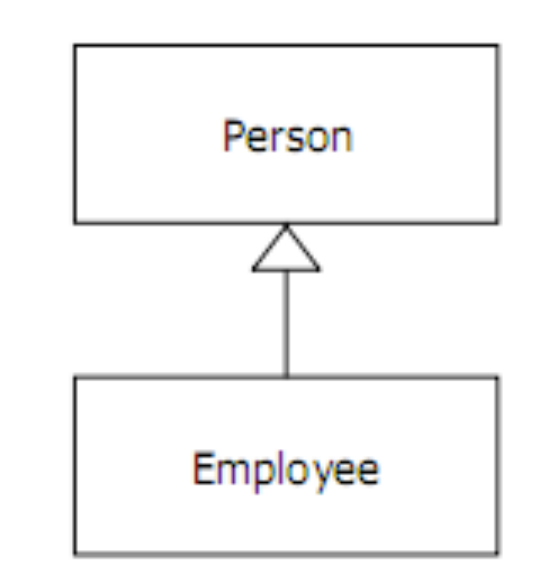


Vers la mise en oeuvre des classes

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

Opération : terme générique désignant le plus souvent des méthodes

Généralisation

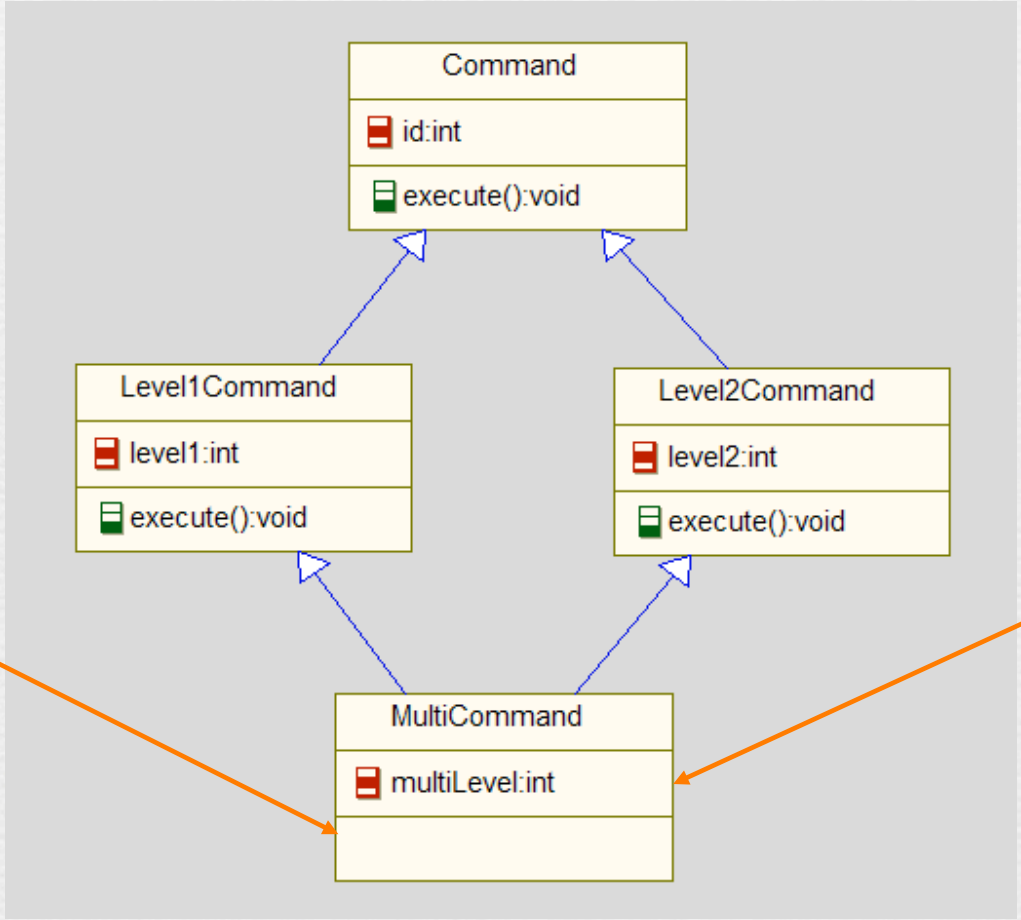



```
class Person
{
}

class Employee extends Person
{
}
```

parlez|uml

Mauvaise généralisation

- Attention à la généralisation multiple:



This is also known as the Diamond of Death. (IBM)

Pattern Composite...