Analyse & conception des systèmes Informatiques.
Diagramme d'objets

DR. Sofiane AOUAG
Université De Batna
Faculté MI– Département Informatique

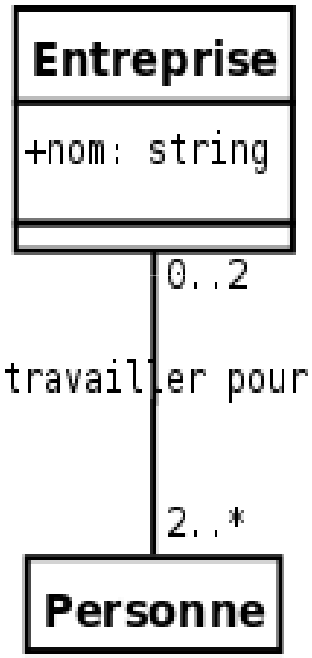
Plan du cours

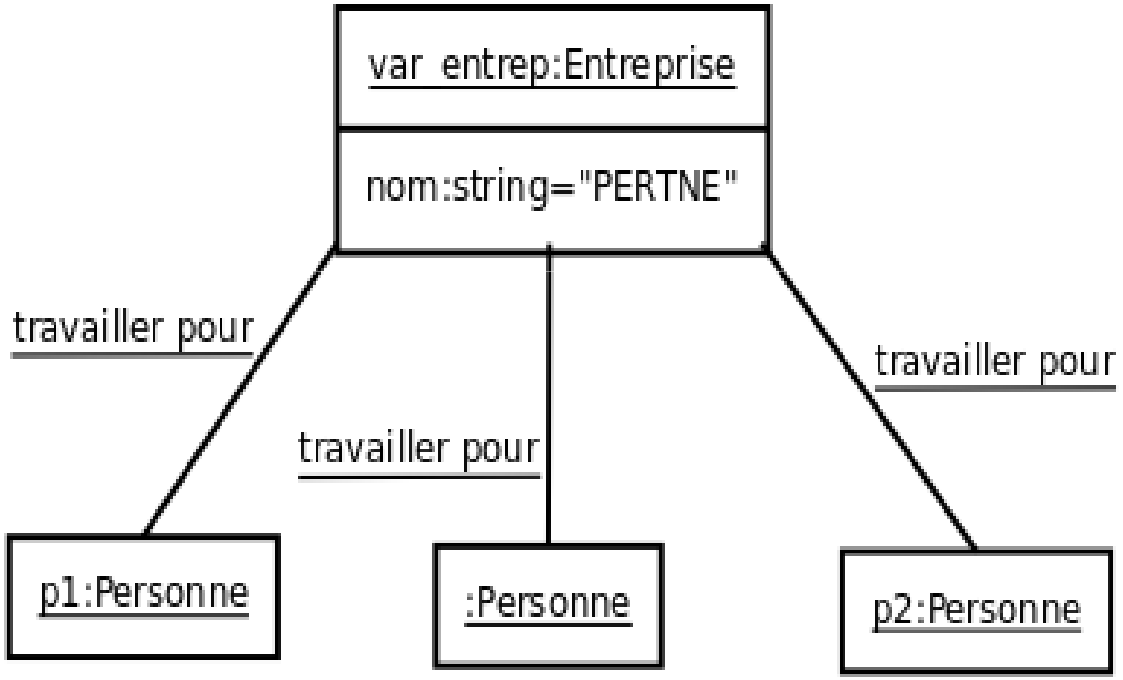
- Introduction
- Éléments graphiques d'un diagramme d'objets
- Formalisme d'un diagramme d'objets
  - Constructeur et destructeur d'objets
  - Durée de vie d'un Objet
  - Objet Composite
  - Les contraintes dans le diagramme d'objets
  - Le langage de contrainte OCL
- Implémentation en java
- Conclusion

Introduction

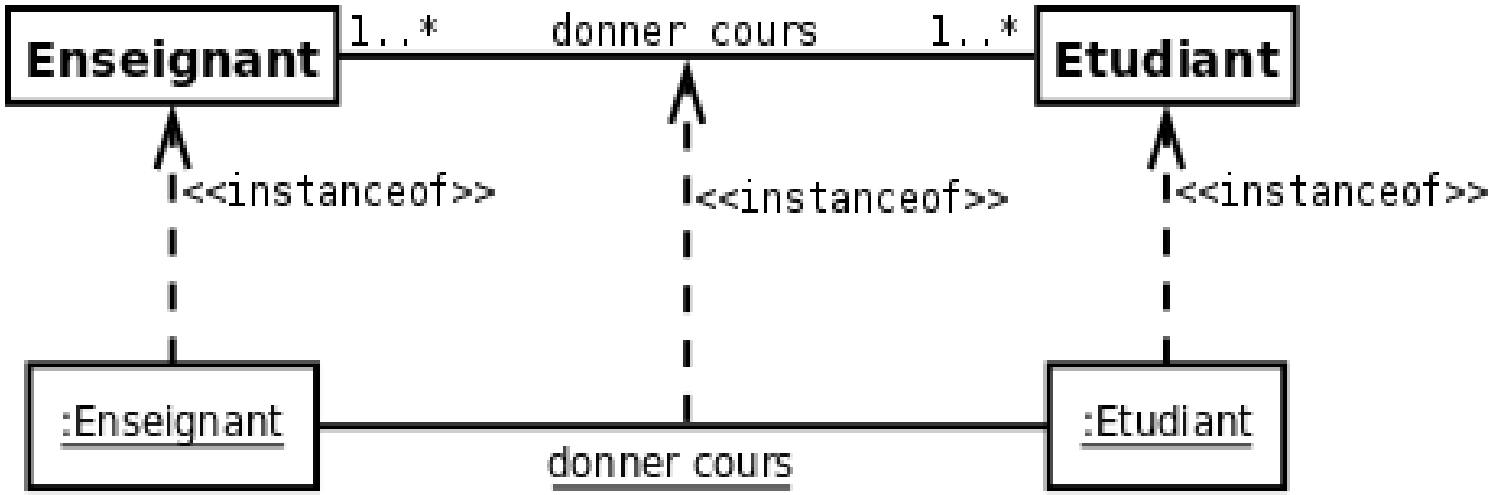
- Le diagramme d'objets permet la représentation d'une instanciation du diagramme de classes en représentant les objets (i.e. instances de classes) et leurs liens (i.e. instances de relations).
- Un diagramme d'objets peut être utilisé pour illustrer le modèle de classes en montrant un exemple qui explique le modèle en précisant certains aspects du système. Il
- Le diagramme de classes modélise les règles et le diagramme d'objets modélise des faits. Il permet de donner une vue figée de l'état d'un système à un instant donné.

Elements graphiques d'un diagramme d'objet





Éléments graphiques d'un diagramme d'objets



Formalisme de diagramme d'objet

Constructeurs et destructeurs d'objets

- Un constructeur est une méthode d'instanciation et d'initialisation d'objets dans la mémoire.
- Le destructeur est une méthode de finalisation et de destruction de l'objet dans la mémoire. Toute classe a un constructeur et un destructeur par défaut, fournis par le compilateur.
- La redéfinition des constructeurs et destructeurs permet de gérer certaines actions qui doivent avoir lieu lors de la création d'un objet et de leur destruction.

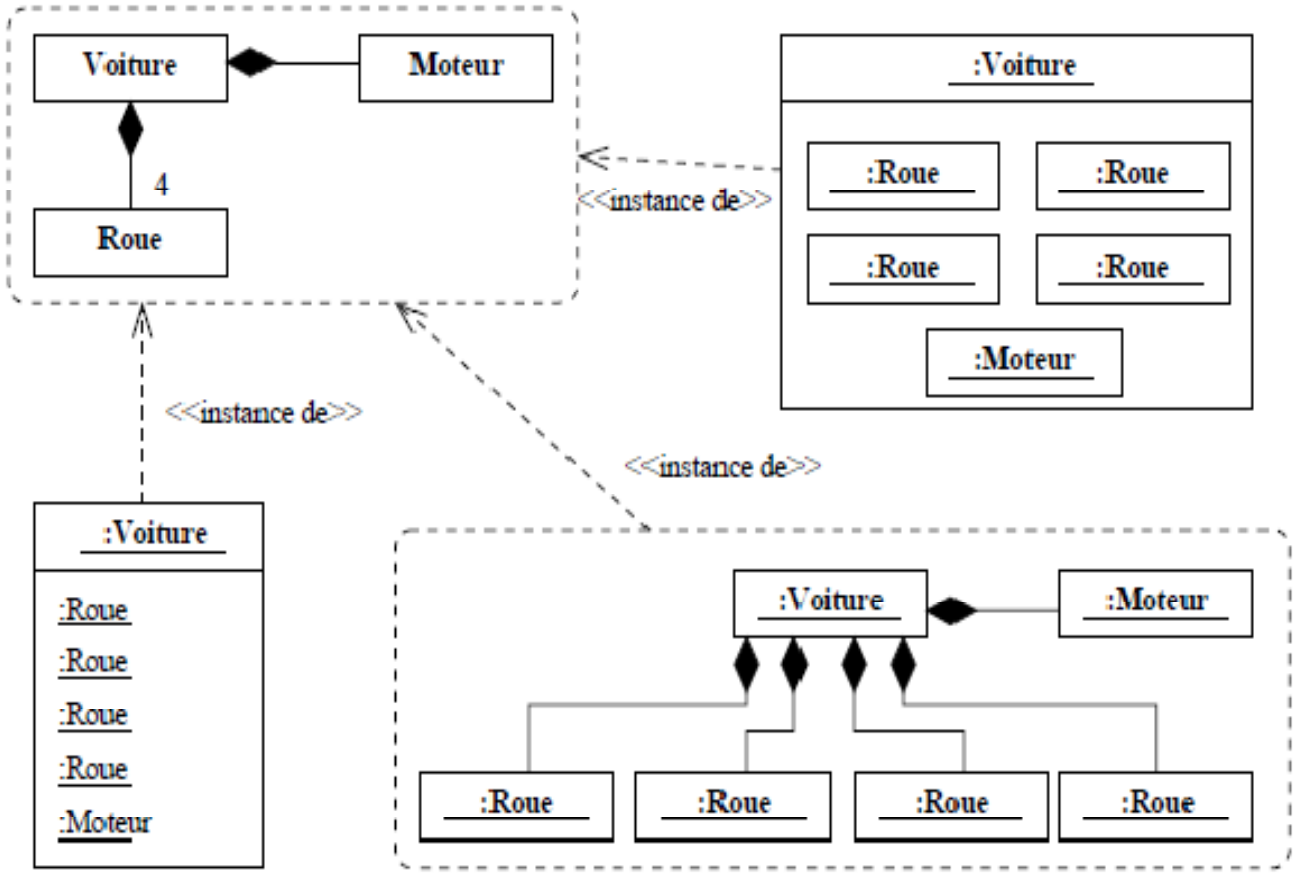
Formalisme de diagramme d'objet

Durée de vie d'un objet

- Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.
- La durée de vie d'un objet passe par trois étapes :
  - la déclaration de l'objet et l'instanciation ;
  - l'utilisation de l'objet en appelant ses méthodes ;
  - la suppression de l'objet : elle se fait en utilisant le destructeur d'objets. Il est à noter qu'en java, cette étape est automatique grâce à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector).

Formalisme de diagramme d'objet

Objet composite



Formalisme de diagramme d'objet

Les contraintes dans le diagramme d'objet

- Une contrainte constitue une condition ou une restriction sémantique exprimée sous forme d'instruction dans un langage textuel qui peut être naturel ou formel.
- Une contrainte désigne une restriction qui doit être appliquée par une implémentation correcte du système.
- On représente une contrainte sous la forme d'une chaîne de texte placée entre accolades ({}).
- Une contrainte possède un nom, on présente celui-ci sous forme d'une chaîne suivie d'un double point (:), le tout précédant le texte de la contrainte.

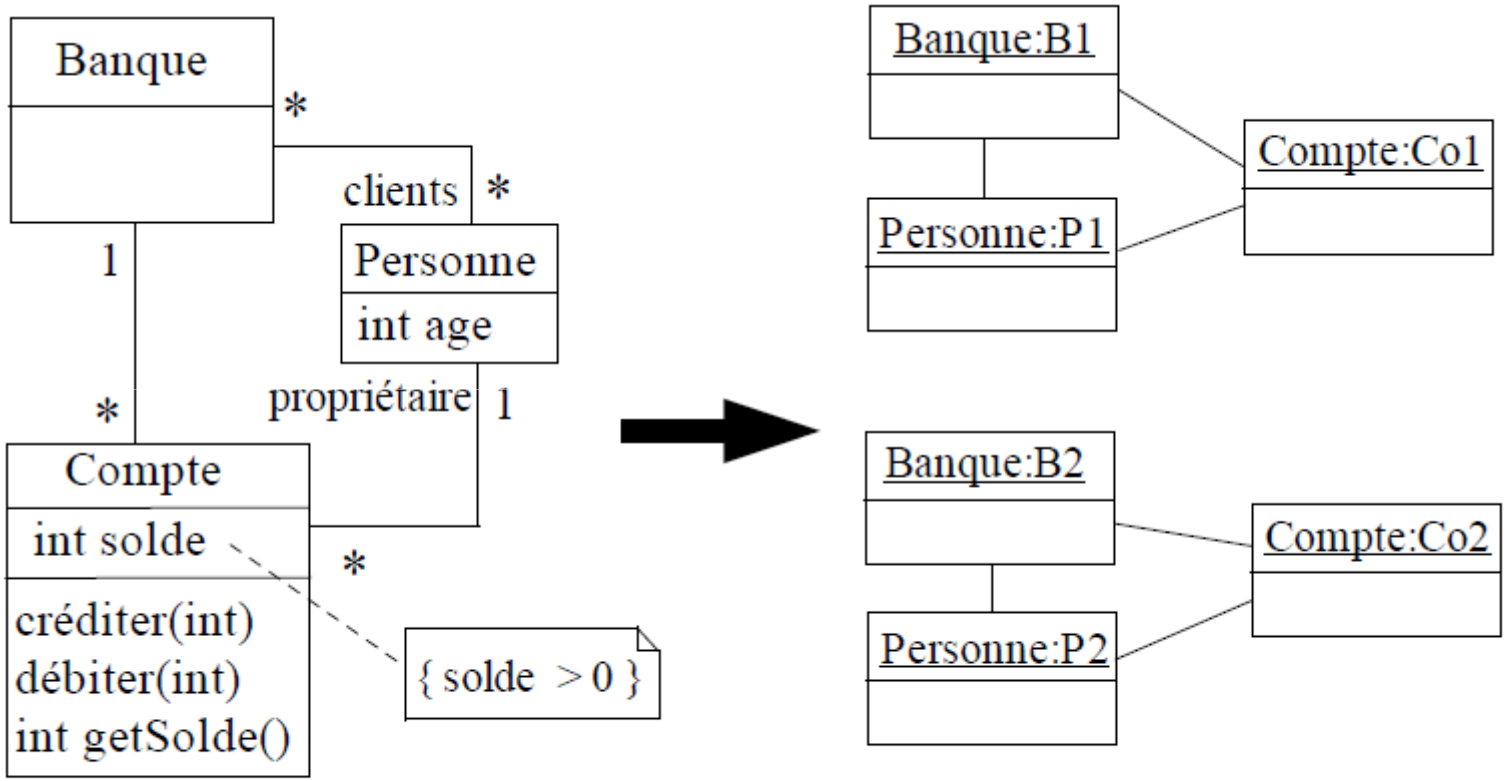
Formalisme de diagramme d'objet

Langage OCL

- OCL (Object Constraint Language) formalise l'expression des contraintes avec les diagrammes d'UML, et en particulier au diagramme de classes.
- OCL existe depuis la version 1.1 d'UML et est une contribution d'IBM. OCL fait partie intégrante de la norme UML depuis la version 1.3 d'UML.
- Dans le cadre d'UML 2.0, les spécifications du langage OCL figurent dans un document indépendant de la norme d'UML, décrivant en détail la syntaxe formelle et la façon d'utiliser ce langage.

Formalisme de diagramme d'objet

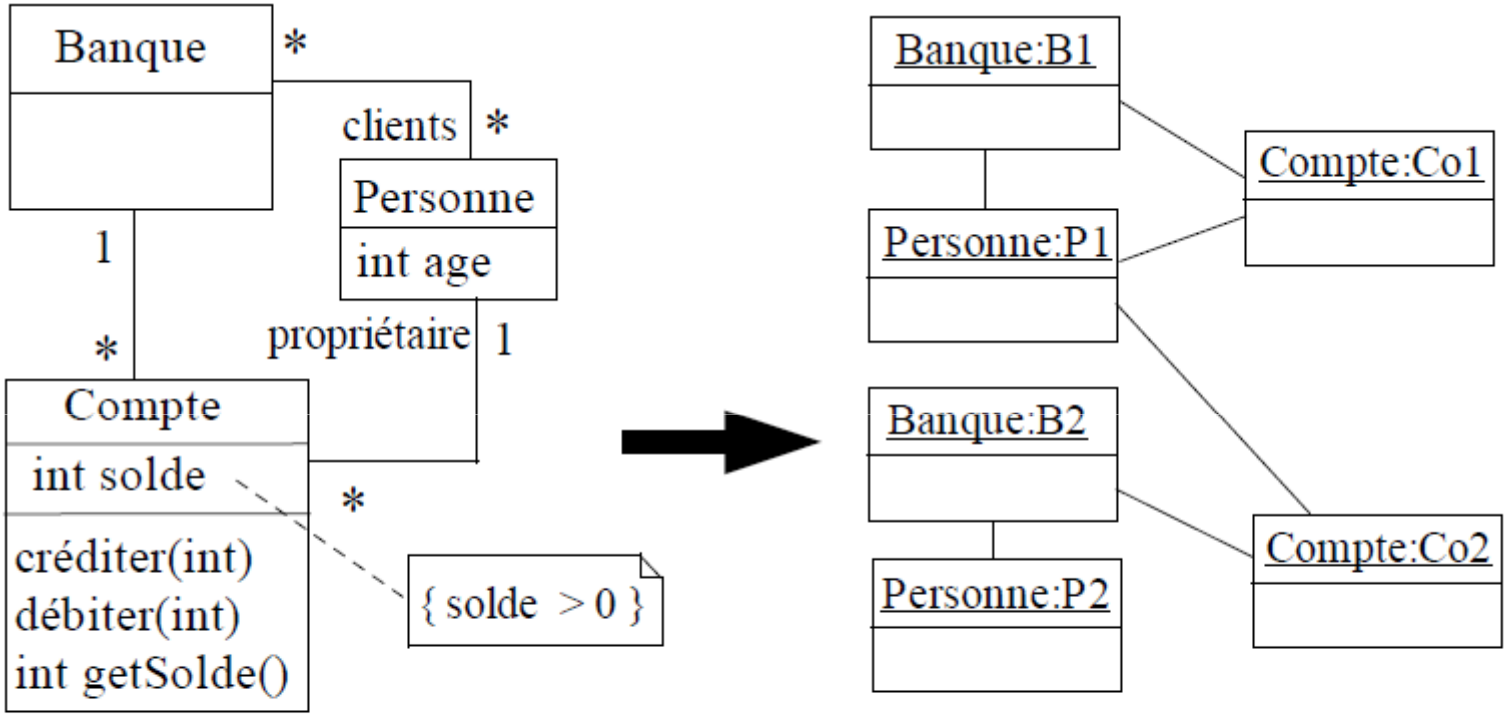
Langage OCL



- Diagramme d'instances valide vis-à-vis du diagramme de classe et de la spécification attendue

Formalisme de diagramme d'objet

Langage OCL



- Diagramme d'instances valide vis-à-vis du diagramme de classe mais ne respecte pas la spécification attendue :
  - Une personne a un compte dans une banque où elle n'est pas cliente
  - Une personne est cliente d'une banque mais sans y avoir de compte

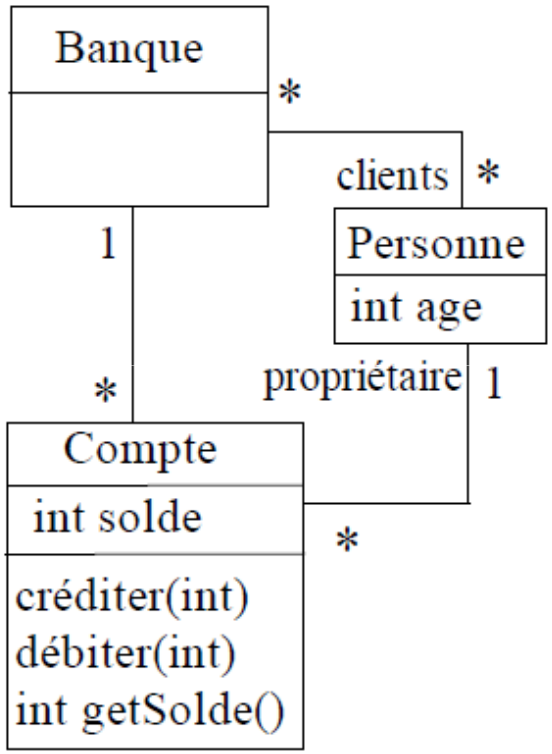
Formalisme de diagramme d'objet

Langage OCL

- OCL peut s'appliquer sur la plupart des diagrammes d'UML et permet de spécifier des contraintes sur l'état d'un objet ou d'un ensemble d'objets comme :
  - les invariants doivent être respectés en permanence;
  - les préconditions doivent être vérifiées avant l'exécution,
  - les postconditions doivent être vérifiées après l'exécution ;
  - des ensembles d'objets destinataires pour un envoi de message ;
  - une expression OCL décrit une contrainte à respecter et pas le « code » d'une méthode

Formalisme de diagramme d'objet

Langage OCL



context Compte

inv: solde > 0

context Compte : débiter(somme : int)

pre: somme > 0

post: solde = solde@pre - somme

context Compte

inv: banque.clients -> includes (propriétaire)

- Avantage d'OCL : langage formel permettant de préciser clairement de la sémantique sur les modèles UML

Formalisme de diagramme d'objet

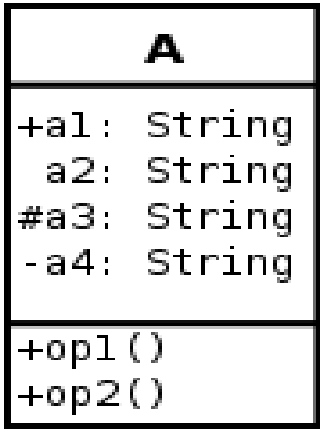
Langage OCL

OCL propose un ensemble de primitives utilisables sur les ensembles :

- `size()` : retourne le nombre d'éléments de l'ensemble
- `isEmpty()` : retourne vrai si l'ensemble est vide
- `notEmpty()` : retourne vrai si l'ensemble n'est pas vide
- `includes(obj)` : vrai si l'ensemble inclut l'objet `obj`
- `excludes(obj)` : vrai si l'ensemble n'inclut pas l'objet `obj`
- `including(obj)` : l'ensemble référencé doit être cet ensemble en incluant l'objet `obj`
- `excluding(obj)` : idem mais en excluant l'objet `obj`
- `includesAll(ens)` : l'ensemble contient tous les éléments de l'ensemble `ens`
- `excludesAll(ens)` : l'ensemble ne contient aucun des éléments de l'ensemble `ens`

Syntaxe d'utilisation : `objetOuCollection -> primitive`

Implémentation en java



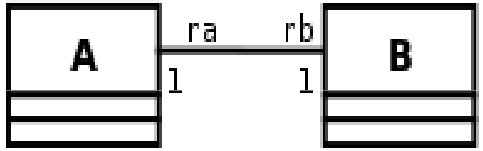

```
public class A {
   public    String a1;
   package   String a2;
   protected String a3;
   private   String a4;
   public void op1() {
      ...
   }
   public void op2() {
      ...
   }
}
```

Implémentation en java

Relation bi-directionnelle 1-1

```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ){
      if ( b.getA() != null ) {    // si b est déjà connecté à un autre A
        b.getA().setB(null);     // cet autre A doit se déconnecter
      }
           this.setB( b );    b.setA( this );
    }
  }
  public B getB() { return( rb ); }
  public void setB( B b ) { this.rb=b; }
}

public class B {
  private A ra;
  public void addA( A a ) {
    if( a != null ) {
      if (a.getB() != null) {    // si a est déjà connecté à un autre B
        a.getB().setA( null );  // cet autre B doit se déconnecter
      }
      this.setA( a ); a.setB( this );
    }
  }
  public void setA(A a){ this.ra=a; }
  public A getA(){ return(ra); }
}
```

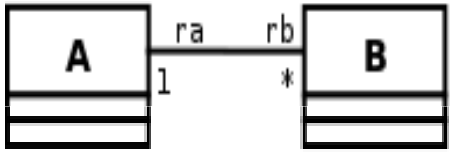


Implémentation en java

Relation bi-directionnelle 1-N

```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public ArrayList <B> getArray() {return(rb);}
  public void remove(B b){rb.remove(b);}
  public void addB(B b){
    if( !rb.contains(b) ){
      if (b.getA()!=null) b.getA().remove(b);
      b.setA(this);
      rb.add(b);    }  }}

public class B {
  private  A ra;
  public B() {}
  public A getA() { return (ra); }
  public void setA(A a){ this.ra=a; }
  public void addA(A a){
    if( a != null ) {
      if( !a.getArray().contains(this)) {
        if (ra != null) ra.remove(this);
        this.setA(a);
        ra.getArray().add(this); } }  }
}
```

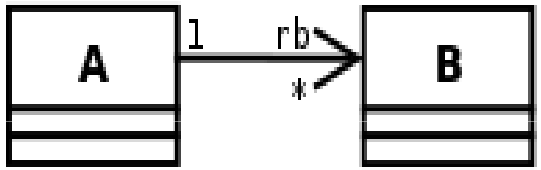


Implémentation en java :

Relation uni-directionnelle 1-N

```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public void addB(B b){
    if( !rb.contains( b ) ) {
      rb.add(b);
    }
  }
}

public class B {
 ... // B ne connaît pas l'existence de A
}
```

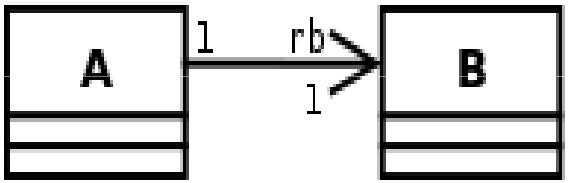


Implémentation en java :

Relation uni-directionnelle 1-1

```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ) {
      this.rb=b;
    }
  }
}

public class B {
 ... // La classe B ne connaît pas
l'existence de la classe A
}
```



Conlusion

- Un diagramme d'objets représente un état du système en utilisant l'ensemble des contraintes prédéfinies. Ces contraintes doivent être formulées en utilisant le langage OCL. Nous avons montré que l'utilisation d'OCL permet d'organiser le travail.
- L'implémentation du diagramme de classes permet présenter des exemples de contenu d'implémentation facilitant la compréhension du diagramme d'objets et son utilisation dans le système.
- Un diagramme d'objets donc ne montre pas l'évolution du système dans le temps. Pour représenter ce fait, en se basant sur les interactions au fil du temps, il faut utiliser le diagramme d'interaction.