Diagramme d'objets (*object diagram*)

1 Présentation

Un diagramme d'objets représente des objets (*i.e.* instances de classes) et leurs liens (*i.e.* instances de relations) pour donner une vue figée de l'état d'un système à un instant donné. Un diagramme d'objets peut être utilisé pour :

- Illustrer le modèle de classes en montrant un exemple qui explique le modèle ;
- Préciser certains aspects du système en mettant en évidence des détails imperceptibles dans le diagramme de classes ;
- Exprimer une exception en modélisant des cas particuliers ou des connaissances non généralisables qui ne sont pas modélisés dans un diagramme de classe ;
- Prendre une image (*snapshot*) d'un système à un moment donné.

Le diagramme de classes modélise les règles et le diagramme d'objets modélise des faits.

Par exemple, le diagramme de classes montre qu'une entreprise emploie au moins deux personnes et qu'une personne travaille dans aux plus deux entreprises. Le diagramme d'objets modélise lui une entreprise particulière (*PERTNE*) qui emploie trois personnes.

Un diagramme d'objets ne montre pas l'évolution du système dans le temps. Pour représenter une interaction, il faut utiliser un diagramme de communication

2 Représentation

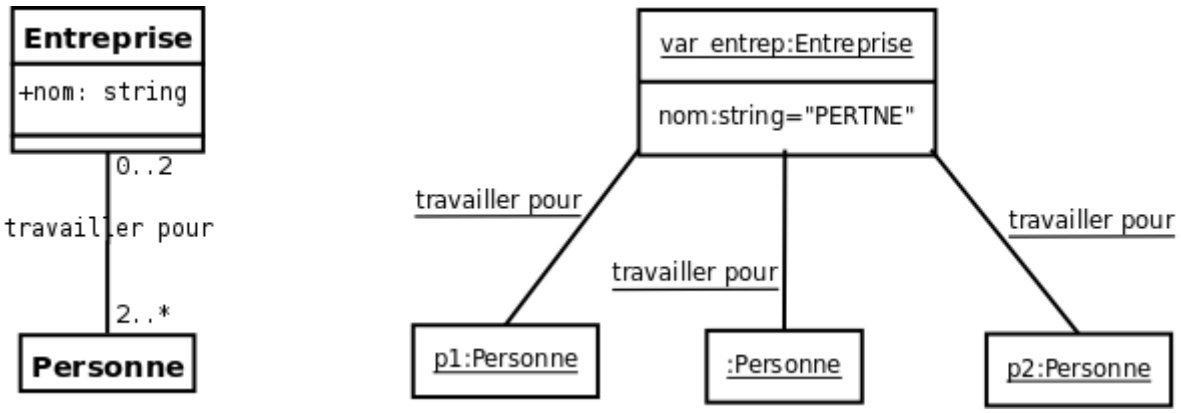


Figure : Exemple de diagramme de classes et de diagramme d'objets associé.

Graphiquement, un objet se représente comme une classe. Cependant, le compartiment des opérations n'est pas utile. De plus, le nom de la classe dont l'objet est une instance est précédé d'un << : >> et est souligné. Pour différencier les objets d'une même classe, leur identifiant peut être ajouté devant le nom de la classe. Enfin les attributs reçoivent des valeurs. Quand certaines valeurs d'attribut d'un objet ne sont pas renseignées, on dit que l'objet est partiellement défini.

Dans un diagramme d'objets, les relations du diagramme de classes deviennent des liens. La relation de généralisation ne possède pas d'instance, elle n'est donc jamais représentée dans un diagramme d'objets. Graphiquement, un lien se représente comme une relation, mais, s'il y a un nom, il est souligné. Naturellement, on ne représente pas les multiplicités.

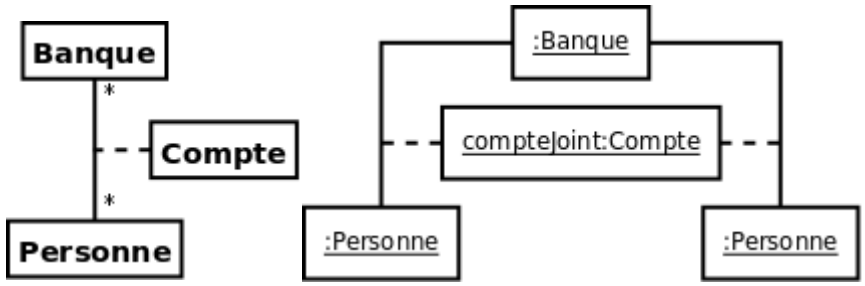


Figure : Le diagramme d'objets de droite, illustrant le cas de figure d'un compte joint, n'est pas une instance *normale* du diagramme de classe de gauche mais peut préciser une situation exceptionnelle.

La norme UML 2.1.1 précise qu'une instance de classe-association ne peut être associée qu'à une instance de chacune des classes associées ce qui interdit d'instancier le diagramme de classe à gauche dans la figure par le diagramme d'objet à droite dans cette même figure. Cependant, un diagramme d'objet peut être utilisé pour exprimer une exception. Sur la figure, le diagramme d'objets à droite peut être légitime s'il vient préciser une situation exceptionnelle non prise en compte par le diagramme de classe représenté à gauche. Néanmoins, le cas des comptes joints n'étant pas si exceptionnel, mieux vaut revoir la modélisation comme préconisé par la figure.

3 Relation de dépendance d'instanciation

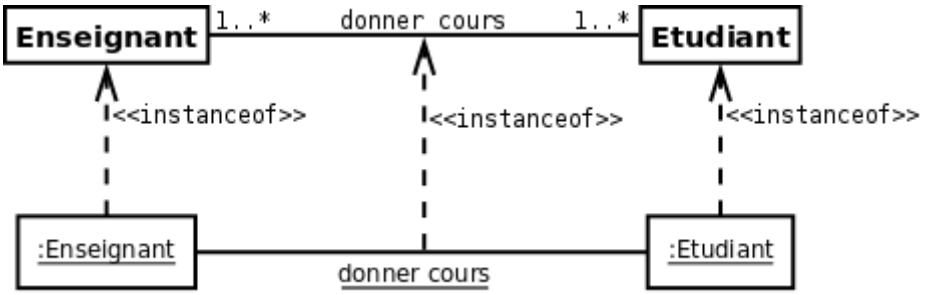


Figure : Dépendance d'instanciation entre les classeurs et leurs instances.

La relation de dépendance d'instanciation (stéréotypée << *instanceof* >>) décrit la relation entre un classeur et ses instances. Elle relie, en particulier, les liens aux associations et les objets aux classes.

Élaboration et implémentation d'un diagramme de classes

1 Élaboration d'un diagramme de classes

Une démarche couramment utilisée pour bâtir un diagramme de classes consiste à :

Trouver les classes du domaine étudié.
Cette étape empirique se fait généralement en collaboration avec un expert du domaine. Les classes correspondent généralement à des concepts ou des substantifs du domaine.

Trouver les associations entre classes.
Les associations correspondent souvent à des verbes, ou des constructions verbales, mettant en relation plusieurs classes, comme << est composé de >>, << pilote >>, << travaille pour >>. *Attention, méfiez vous de certains attributs qui sont en réalité des relations entre classes.*

Trouver les attributs des classes.
Les attributs correspondent souvent à des substantifs, ou des groupes nominaux, tels que << la masse d'une voiture >> ou << le montant d'une transaction >>. Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs. Vous pouvez ajouter des attributs à toutes les étapes du cycle de vie d'un projet (implémentation comprise). N'espérez pas trouver tous les attributs dès la construction du diagramme de classes.

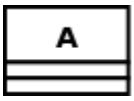
Organiser et simplifier le modèle
en éliminant les classes redondantes et en utilisant l'héritage.

Itérer et raffiner le modèle.
Un modèle est rarement correct dès sa première construction. La modélisation objet est un processus non pas linéaire mais itératif.

2 Implémentation en Java

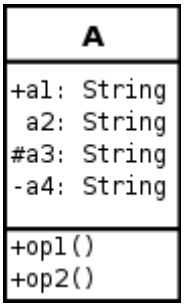
Classe

Parfois, la génération automatique de code produit, pour chaque classe, un constructeur et une méthode finalize comme ci-dessous. Rappelons que cette méthode est invoquée par le *ramasse miettes* lorsque celui-ci constate que l'objet n'est plus référencé. Pour des raisons de simplification, nous ne ferons plus figurer ces opérations dans les sections suivantes.



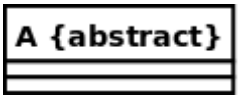

```
public class A {
  public A() {
    ...
  }
  protected void finalize() throws Throwable {
    super.finalize();
    ...
  }
}
```

Classe avec attributs et opérations




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

Classe abstraite

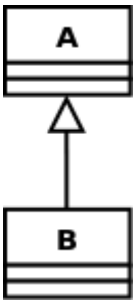



```
public abstract class A {
  ...
}
```

Interface


```
public interface A {
  ...
}
```

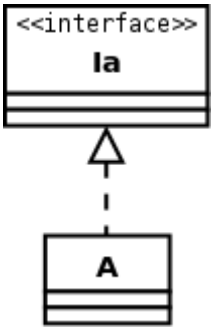
Héritage simple




```
public class A {
  ...
}

public class B extends A {
  ...
}
```

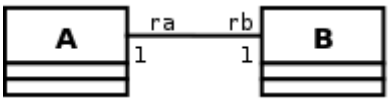
Réalisation d'une interface par une classe




```
public interface Ia {
  ...
}

public class A implements Ia {
  ...
}
```

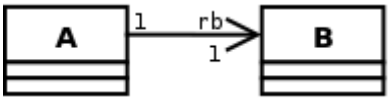
Association bidirectionnelle 1 vers 1




```
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); }
}
```

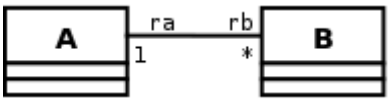
Association unidirectionnelle 1 vers 1




```
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
}
```

Association bidirectionnelle 1 vers N

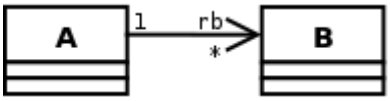



```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public ArrayList <B> getArray() {return(rb);}
  public void remove(B b){rb.remove(b);}
  public void addB(B b){
    if( !rb.contains(b) ){
      if (b.getA()!=null) b.getA().remove(b);
      b.setA(this);
 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

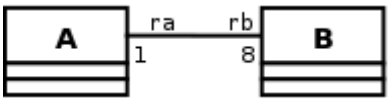
Association unidirectionnelle 1 vers plusieurs




```
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
}
```

Association 1 vers N

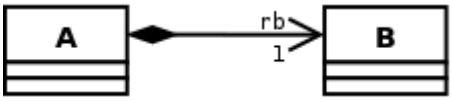


Dans ce cas, il faut utiliser un tableau plutôt qu'un vecteur. La dimension du tableau étant données par la cardinalité de la terminaison d'association.

Agrégations

Les agrégations s'implémentent comme les associations.

Composition



Une composition peut s'implémenter comme une association unidirectionnelle.