Génie Logiciel

Programmation orientée objet et Modélisation UML

Nicolas Delanoue

Université d'Angers - Polytech Angers

En quoi consiste la modélisation ?

Définition

Modéliser, c'est construire une représentation abstraite de la réalité.

En quoi consiste la modélisation ?

Définition

Modéliser, c'est construire une représentation abstraite de la réalité.

Exemples

- Equation de la dynamique de Newton,

$$m\ddot{x}(t) = \sum_i F_i.$$

- Modèle géométrique d'un robot,
- Modèle dynamique d'un robot,
- Equation météorologique (attention aux chaos).

s

FIGURE – Différentes vues, différents modèles.

Pourquoi modéliser en génie logiciel ?

Cela sert à communiquer, afin de

- représenter et construire des systèmes complexes,
- représenter et évaluer les différentes solutions.

Nature de la modélisation

- Informelle : Documents, ...
- Semi-formelle : UML, ...
- Formelle : possibilité de vérifier mathématiquement l'adéquation entre les besoins formalisés et le logiciel réalisé.

Quel langage utiliser pour modéliser ?

- Plusieurs langages existent (plus ou moins proche du langage de codage retenu)
- Un seul a réussi à s'imposer comme le langage pour la programmation orienté objet : UML
  - standardisé par l'Open Management Group,
  - mondialement utilisé (plus de 80% des projets),
  - bien outillé et documenté (livres, tutoriels, mooc, ...)

UML Aujourd'hui

Remarques

- UML est le langage de modélisation orienté objet le plus connu et le plus utilisé,
- UML n'est pas une méthode,
- peu d'utiliseurs connaissent vraiment le standard, ils ont une vision outillé et c'est très déjà très bien,
- UML est critiqué car pas assez formel.

Trois utilisations possibles d'UML

1 - Comme un langage pour faire des croquis, esquisses, ébauches

- Pour échanger, communiquer rapidement une idée,
- Les modèles ne sont pas forcément complets.

Objectifs : Explorer, analyser, réfléchir, décider.
Probablement la manière dont UML est le plus utilisée.

Trois utilisations possibles d'UML

2 - Comme un langage de spécification de modèles, patrons . . .

Décrire complètement ou finement un modèle pour

- coder directement,
- prendre des décisions poussées de conception,
- générer automatiquement du code (classe et signature des méthodes),
- obtenir des modèles via Reverse Engineering afin de casser les dépendances et améliorer la conception.

Objectifs : Concevoir, Pérenniser, Générer du code.
Manière dont UML est utilisé dans de gros projet.

Trois utilisations possibles d'UML

3 - Comme un langage de programmation

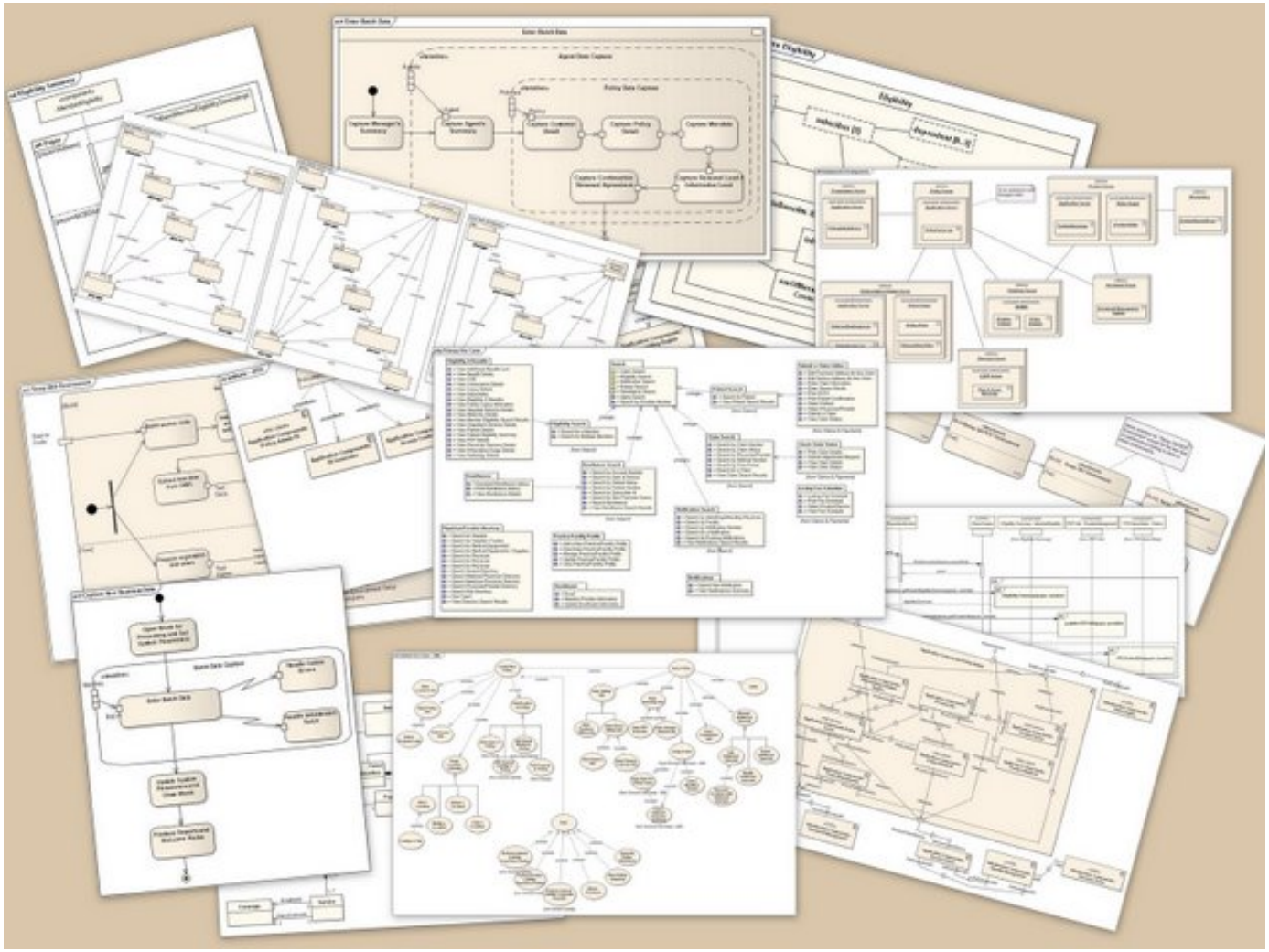
- Description complète, le modèle UML devient la source du code (avec le corps des méthodes).
- Un outillage sophistiqué pas vraiment mature.

Objectifs : Génération automatique de tout le code
Manière dont certains reveurs souhaitent qu'UML devienne.

Les phases de développement couvertes par UML

- Expression des besoins, ok
- Analyse, ok
- Conception, ok
- Réalisation, ok si UML comme langage de programmation,
- Validation,
- Déploiement, ok
- Maintenance.

En UML, les informations sont représentées sous forme de diagrammes :



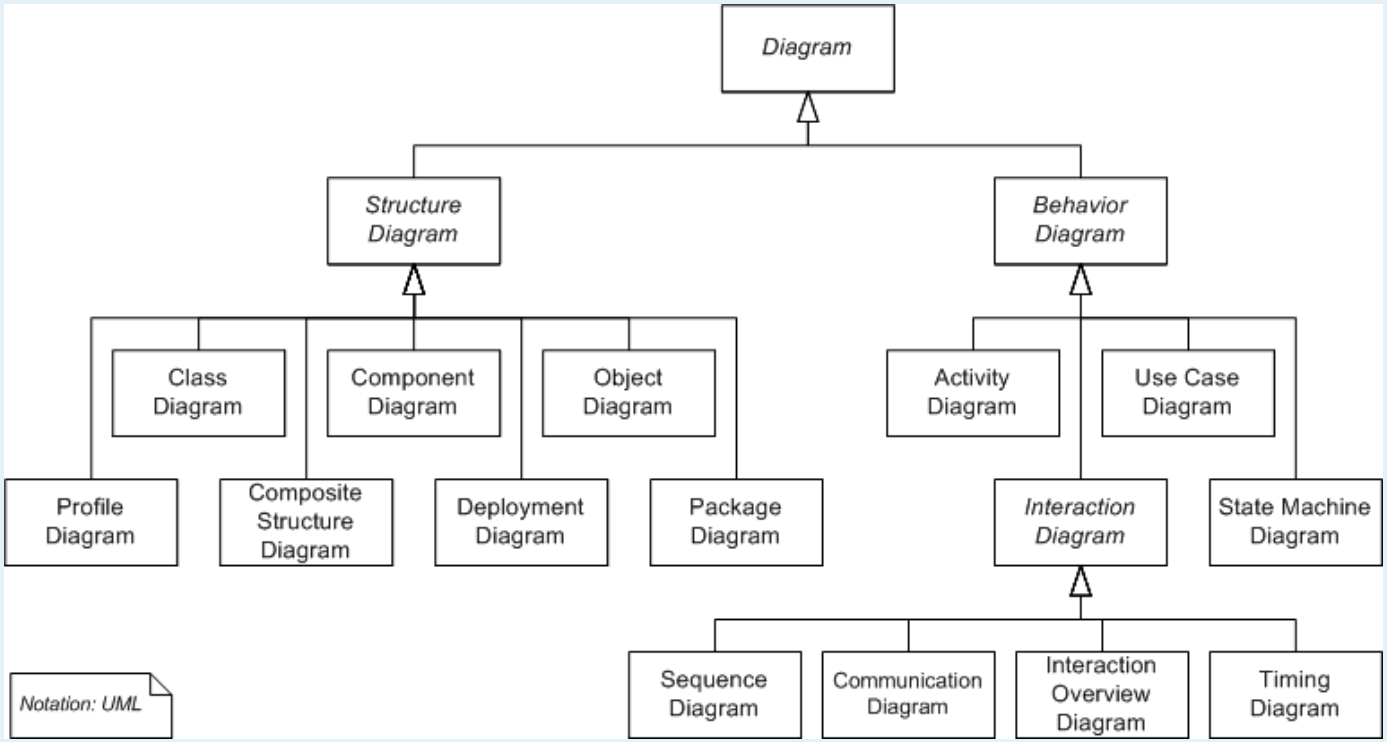


FIGURE – Hiérarchie des diagrammes UML 2.5, montrée comme un diagramme de classes.

Définition

Un *cas d'utilisation* représente un ensemble d'actions qui sont réalisés par le système. Ces actions produisent un résultat observable pour un acteur particulier.

Exemple

Un client retire de l'argent auprès d'un distributeur de monnaie.

- acteur : client,
- cas d'utilisation : retirer de l'argent,
- système : distributeur de monnaie.

Diagramme de cas d'utilisation

Un diagramme des cas d'utilisation modélise :

- des acteurs,
- des cas d'utilisations,
- le système,
- des relations (associations, héritage, et dépendances).

Exemple

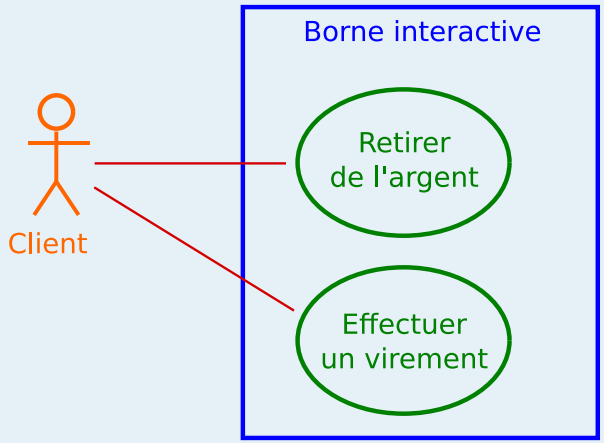
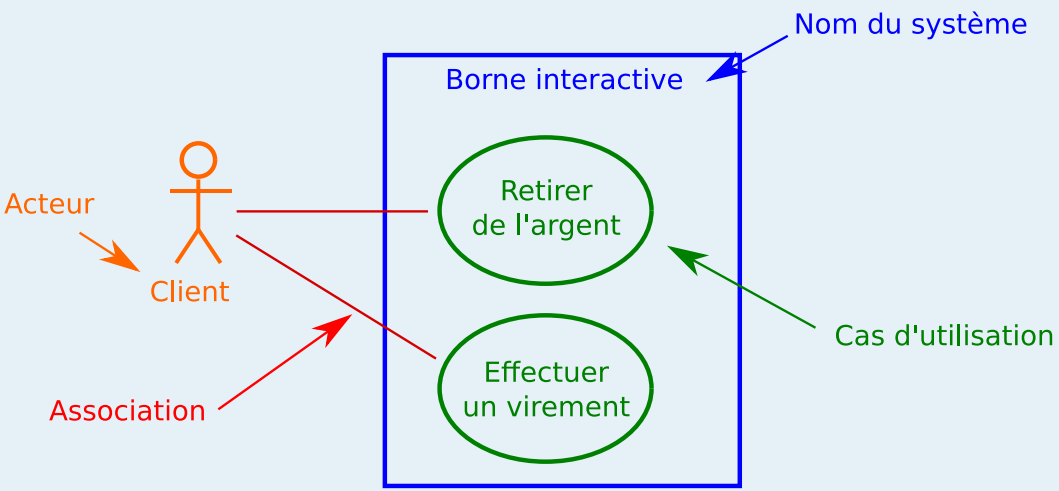


Diagramme de cas d'utilisation

Un diagramme des cas d'utilisation modélise :

- des acteurs,
- des cas d'utilisations,
- le système,
- des relations (associations, héritage, et dépendances).

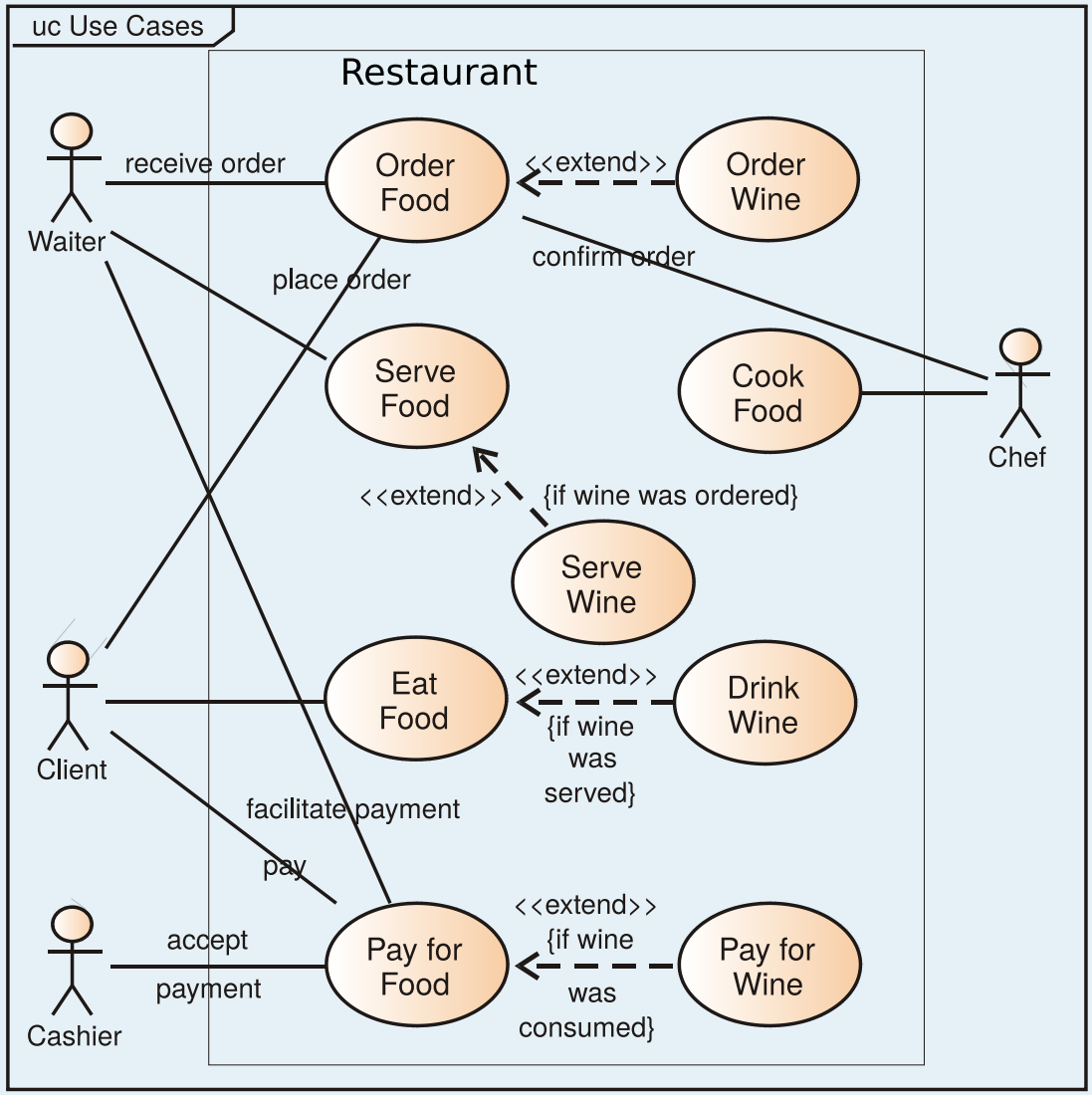
Exemple

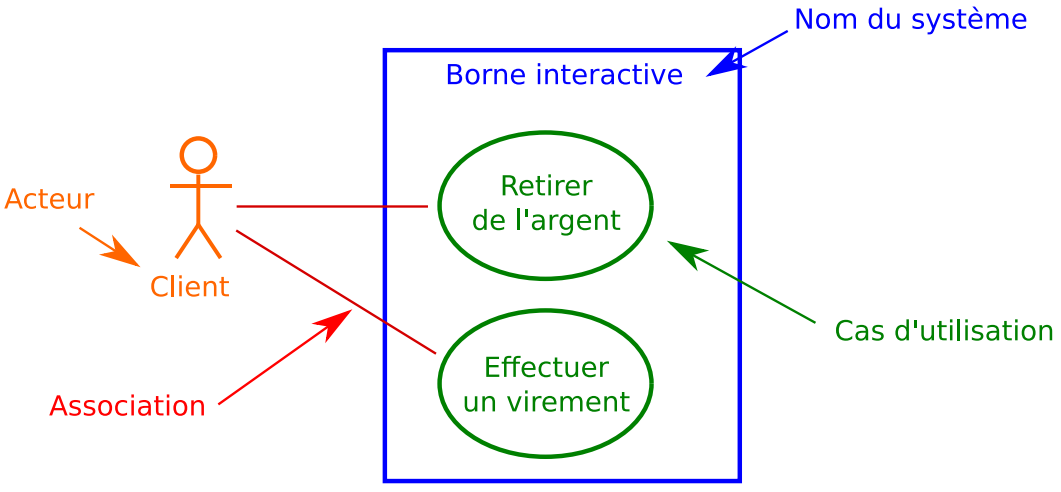


Objectifs de ce diagramme

- Communiquer,
- Capturer les besoins fonctionnels du système,
- Délimiter le système,
- Visualiser un cahier des charges graphiquement.

Exemple de diagramme de cas d'utilisation



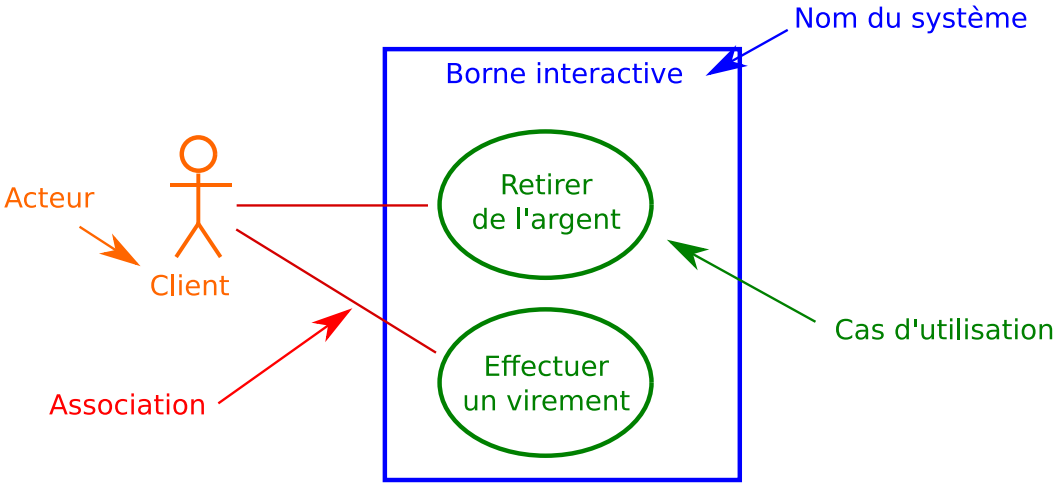


Définition

Un acteur représente un rôle joué par une entité externe qui interagit directement avec le système.

Remarque

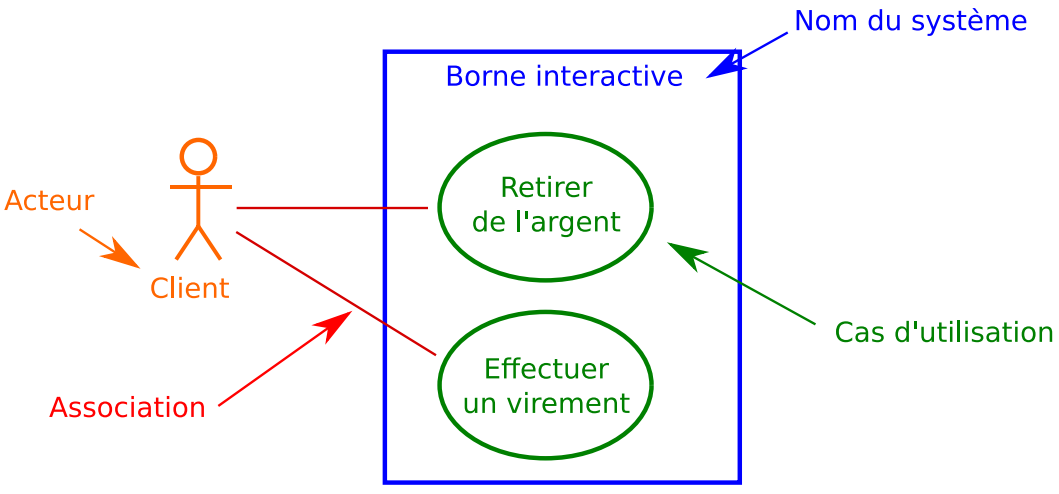
Un acteur peut être un utilisateur, un dispositif matériel comme un imprimante ou encore un autre système (e.g. gestion de base de données).



- Chaque cas d'utilisation spécifie un comportement attendu du système considéré comme un tout, sans imposer le mode de réalisation.
- Synthaxiquement, on utilise l'infinitif pour le nommer.

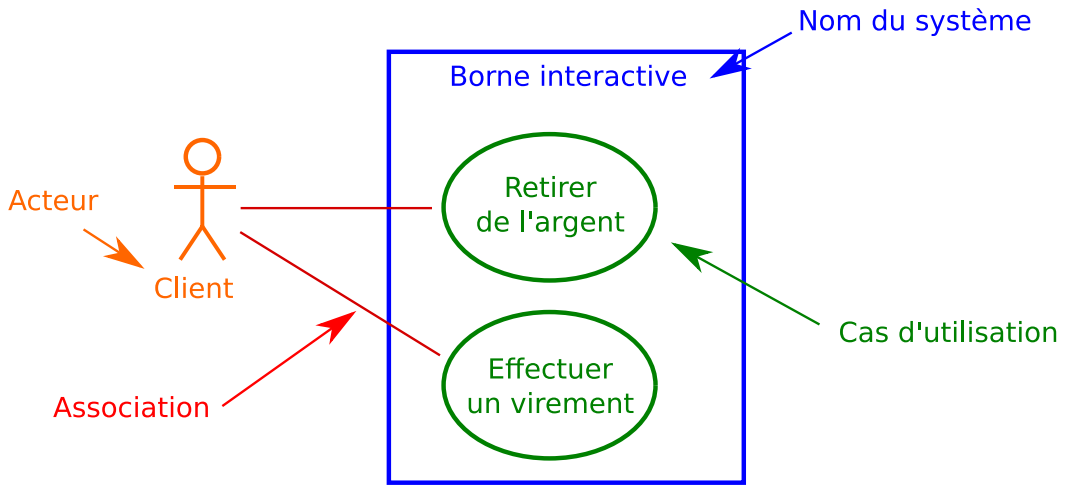
Indentifier les cas d'utilisations

- Quels sont les services rendus par le système ?
- Quelles sont les interactions entre les acteurs et le système ?
- Quels sont les évènements perçus par le système ?



Définition

Le système représente les limites de l'application considérée et regroupe un ensemble de cas d'utilisation.



Les relations

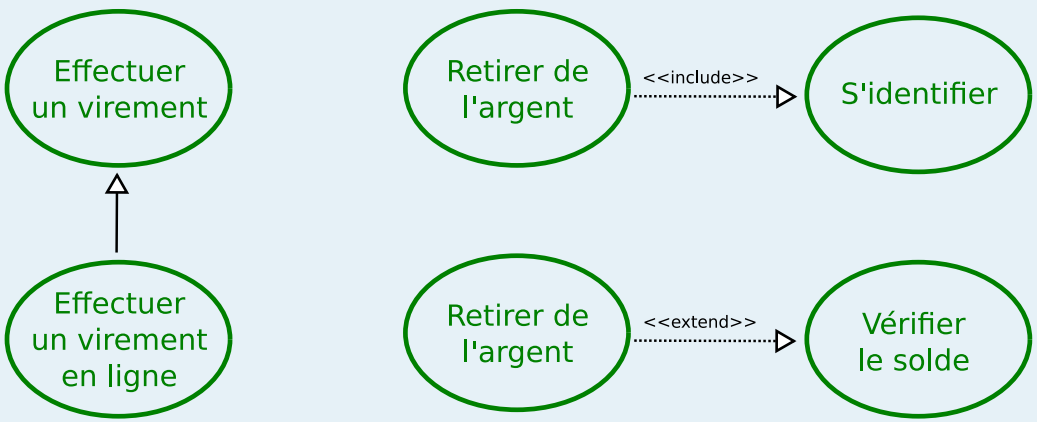
Les relations peuvent exister soit :

- entre les acteurs (on parle d'héritage),
- entre un acteur et un cas d'utilisation,
- entre deux cas d'utilisations.

Entre deux cas d'utilisations, il existe trois type de relations :

- Généralisation (héritage) : le cas fils spécialise le cas père.
- Inclusion(«include») le cas source incorpore nécessairement le cas cible,
- Extension («extend») : le cas de base incorpore éventuellement le cas cible.

Exemple



Résumé

Les diagrammes de cas d'utilisation

- permettent de décrire le système de façon très abstraite,
- offrent une vue fonctionnelle (décrivent le "quoi" et non pas le "comment".),
- sont simples et peuvent être complétés avec
  - des diagrammes de séquences,
  - des diagrammes d'activités.

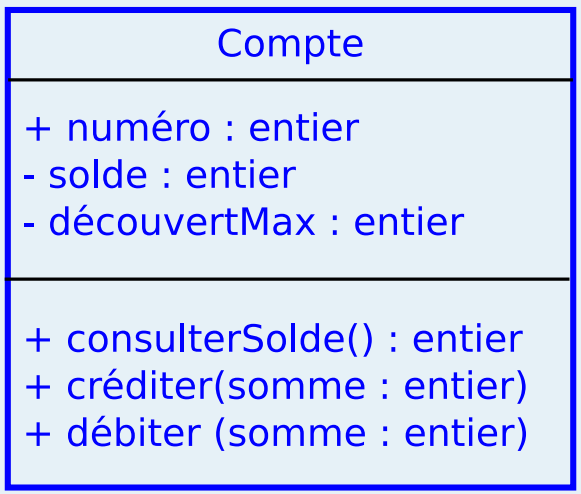
  pour décrire les scénarii.

Définition - Le diagramme de classe

Une classe est représentée par un rectangle séparé en trois parties :

- la première partie contient le nom de la classe
- la seconde contient les attributs de la classe
- la dernière contient les méthodes de la classe

Exemple



Visibilité

+ accès public, toutes les autres classes ont accès à cet attribut.

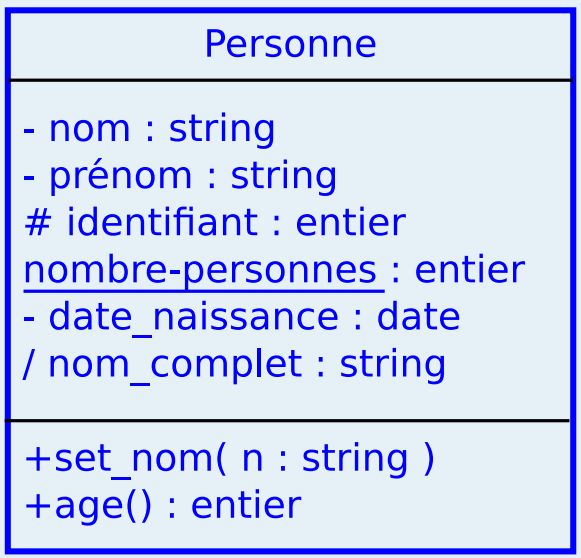
accès protégé, seules la classe elle-même et les classes filles (héritage) ont accès à cet attribut.

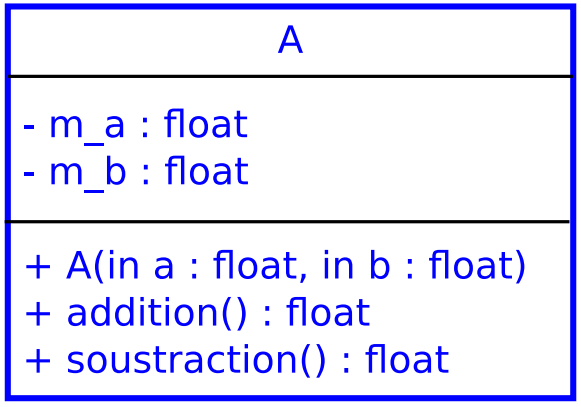
- accès privé, seule la classe elle-même a accès à cet attribut.

Remarques

- UML définit son propre ensemble de types : Integer, Real, String, ...
- Un attribut ou une méthode peut être un attribut de classe, il est alors souligné.
- Un attribut peut être calculé à partir des autres, il est alors précédé de "/".

Exemple plus détaillé






```
public class A {
// Attributs
private float m_a ;
private float m_b ;
// Methodes
public A(float a, float b) { m_a = a; m_b = b; }
public float addition() {return m_a+m_b;}
public float soustraction() {return m_a-m_b;}
public void set_a(float value) {m_a=value;}
public void set_b(float value) {m_b=value;}
}
```

Diagramme de classes 1

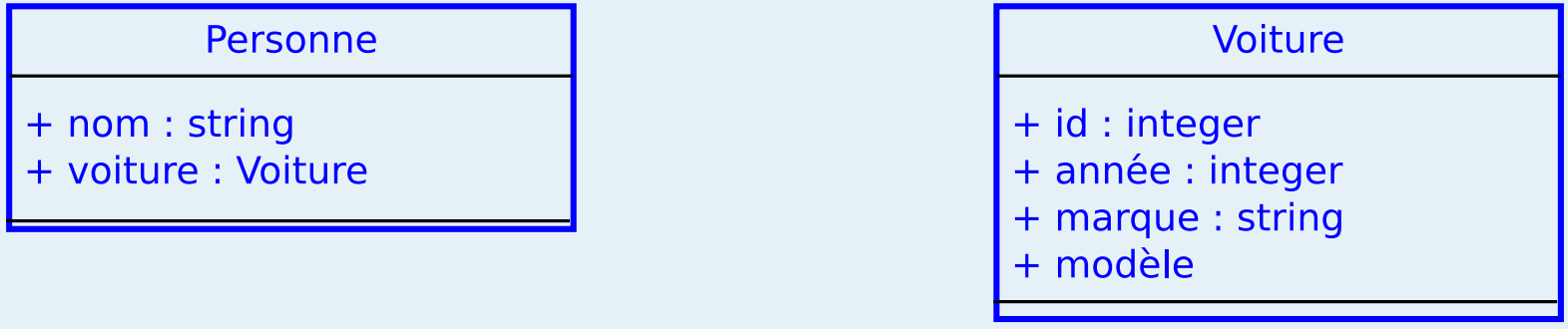
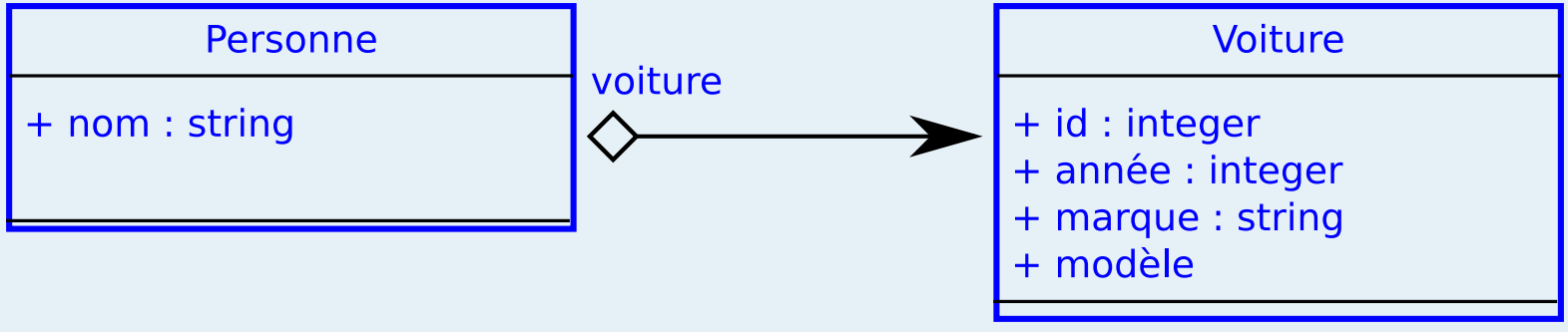


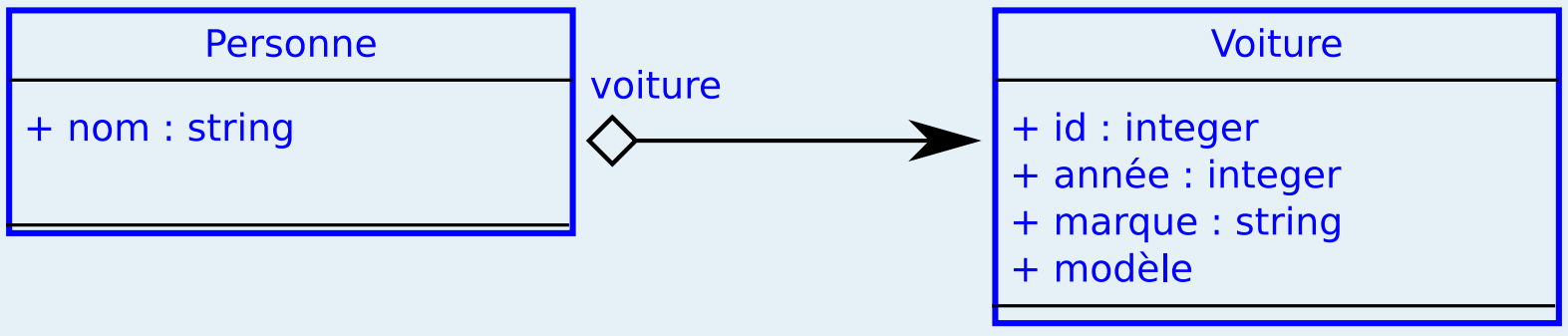
Diagramme de classes 2



Définition

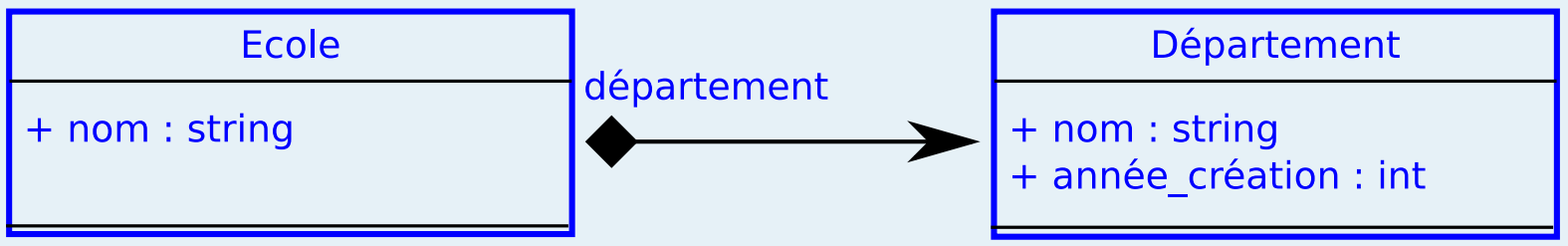
Les deux diagrammes précédents sont équivalents.

Agrégation



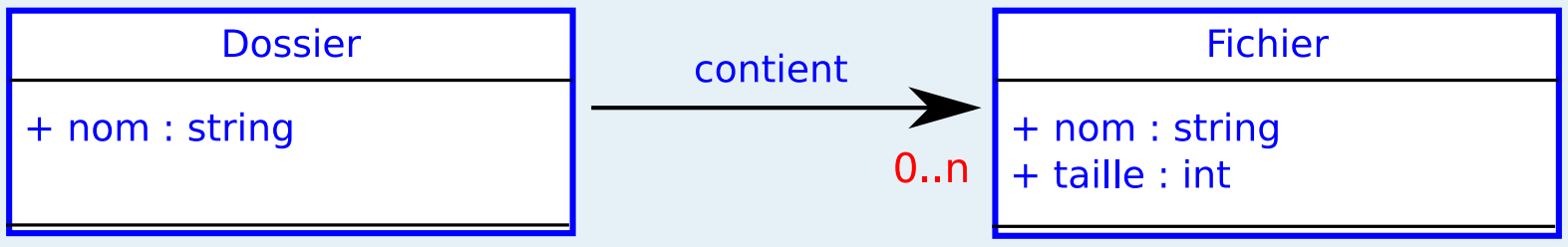
- L'agrégation exprime une composition faible (partagée),
- Un composant peut faire partie de plusieurs composites,
- le cycle de vie du composant est indépendant de celui du composite.

Composition

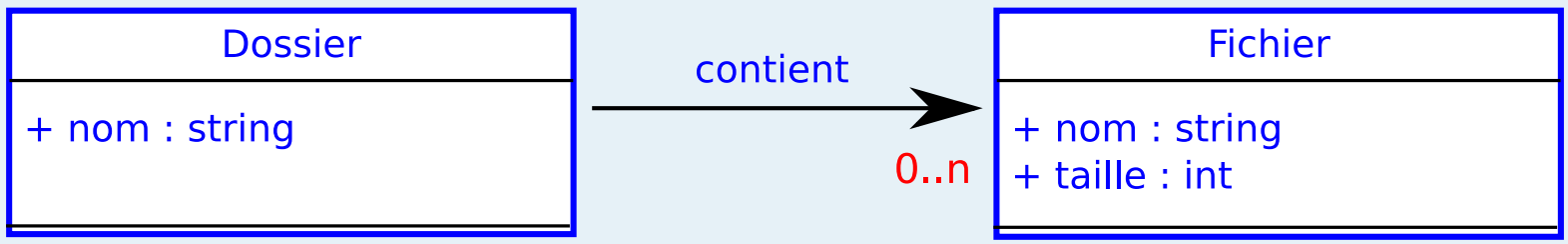


- La composition exprime une composition forte (n'est pas partagée),
- Un composant peut faire partie que d'un seul composite,
- leur cycle de vie sont liés : si le composite est détruit alors les composants aussi.

Cardinalité et navigabilité



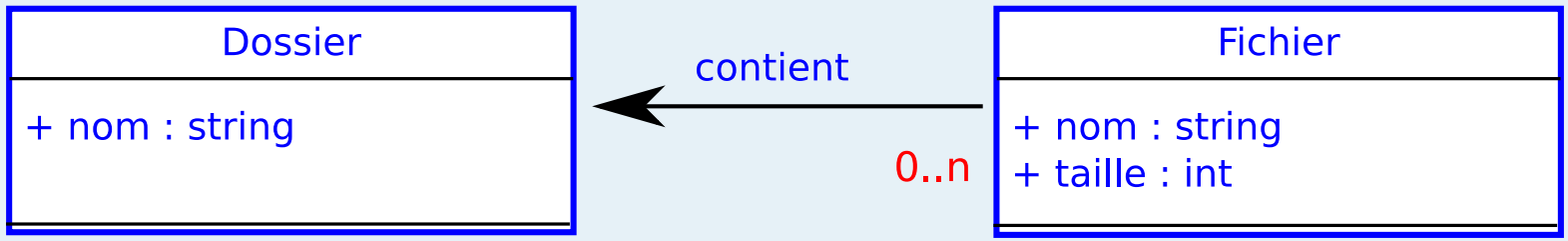
Cardinalité et navigabilité



Code Java

```
public class Dossier
{
  public String nom ;
  public ArrayList<Fichier> fichiers;
}
```

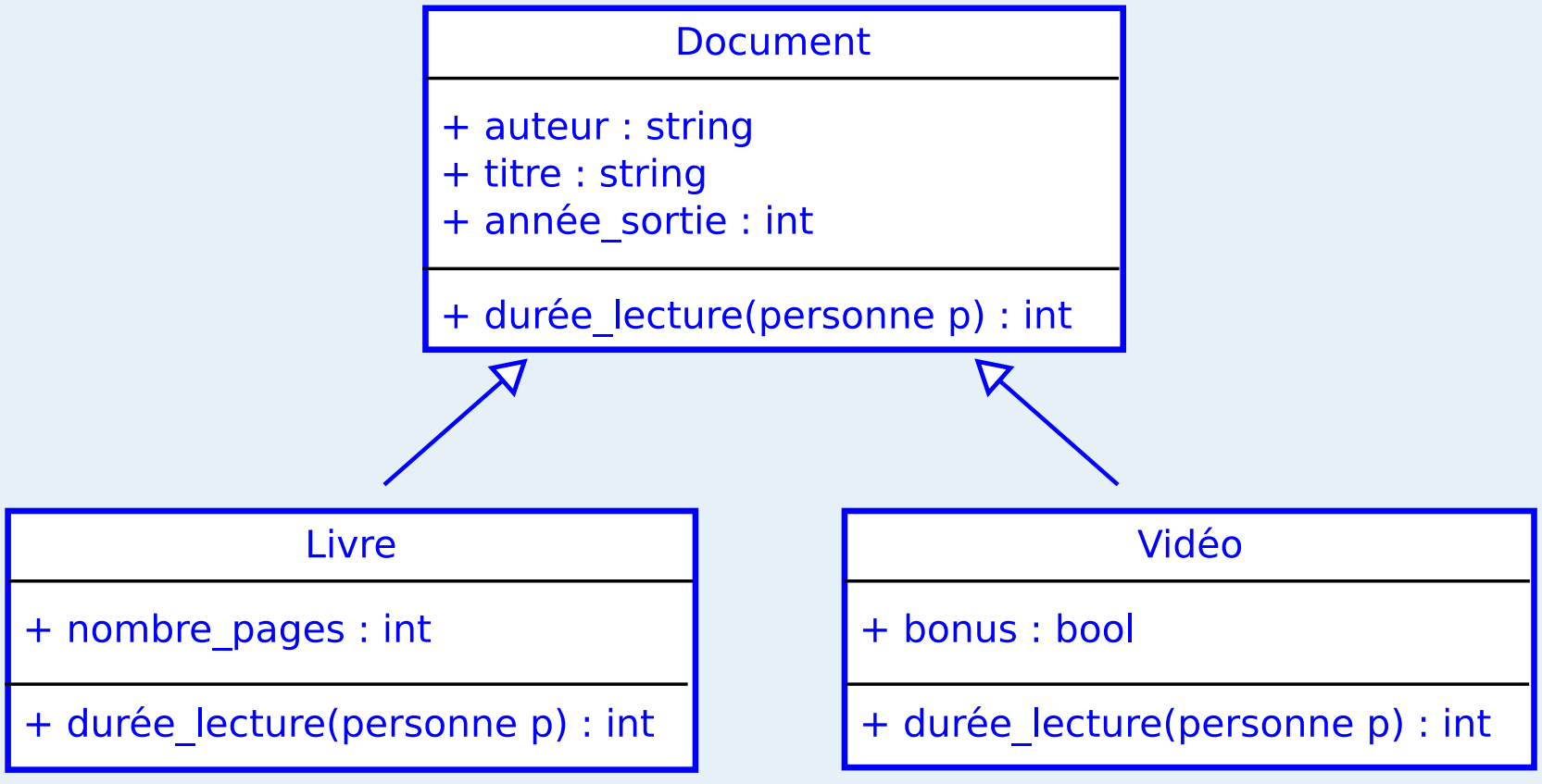
Cardinalité et navigabilité



Code Java

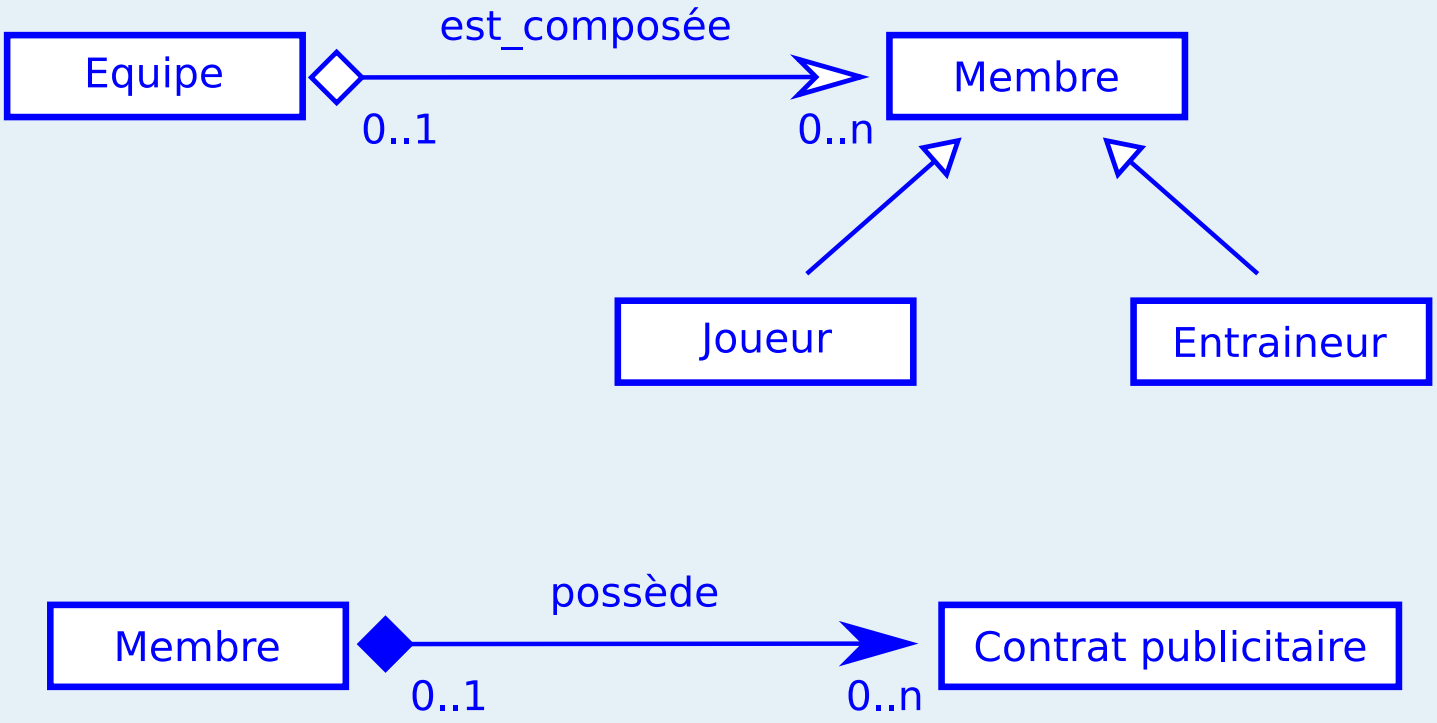
```
public class Fichier
{
  public String nom ;
  public Integer taille;
  public Dossier repertoire ;
}
```

Héritage



Les classes “filles” réutilisent du code déjà écrit dans la classe mère.
La méthode `durée_lecture` est spécialisée dans les sous-classes.

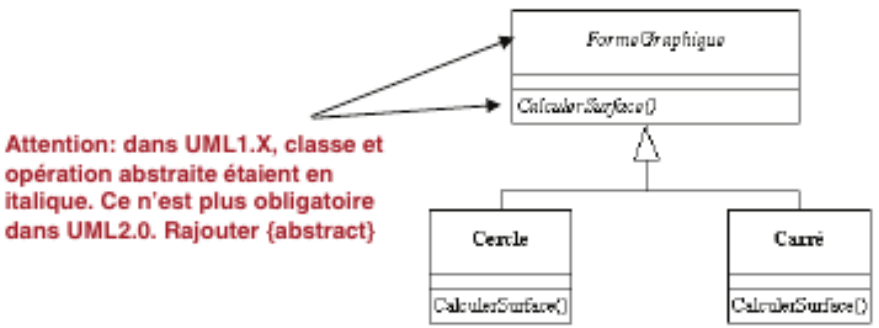
Héritage implicite



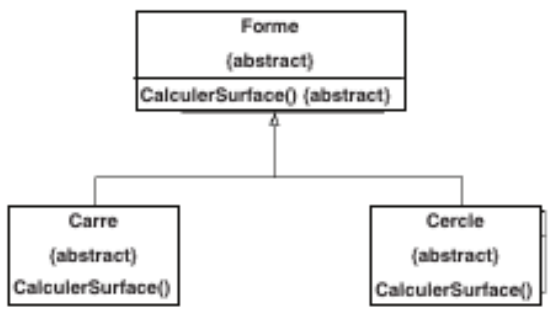
L'héritage des associations est implicite.

Opérations abstraites

- Une classe abstraite est une classe qui contient au moins une opération abstraite
  - Capture des comportements communs
  - Servent à structurer le système
  - Ne peut pas être instanciée
- Une opération abstraite est une opération dont l'implémentation est laissée aux classes filles



Attention: dans UML1.X, classe et opération abstraite étaient en italique. Ce n'est plus obligatoire dans UML2.0. Rajouter {abstract}



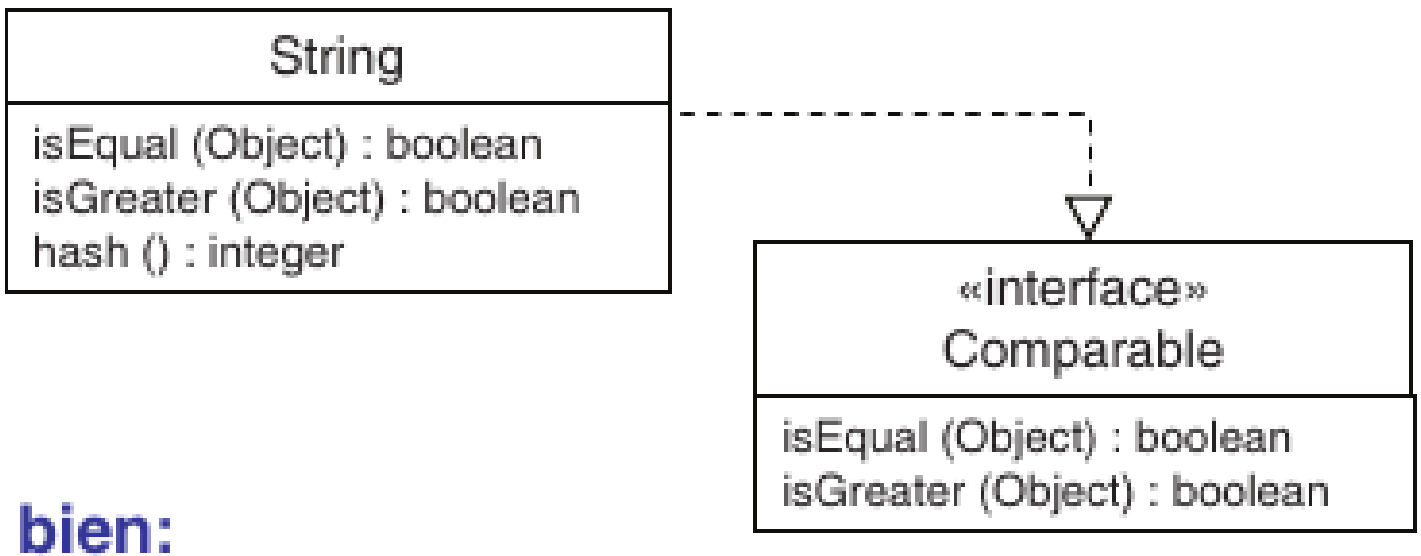
Interfaces

Définition

Une interface est un ensemble de signatures de méthode sans corps.

Remarques

- Peut-être vu comme une classe abstraite dont toutes les opérations sont abstraites,
- Puissant moyen de typage,
- Une classe peut réaliser une ou plusieurs interfaces,
- Un contrat qui engage la classe qui réalise l'interface à implémenter ces méthodes.


String
isEqual (Object) : boolean
isGreater (Object) : boolean
hash () : integer
«interface»
Comparable
isEqual (Object) : boolean
isGreater (Object) : boolean

)u bien:

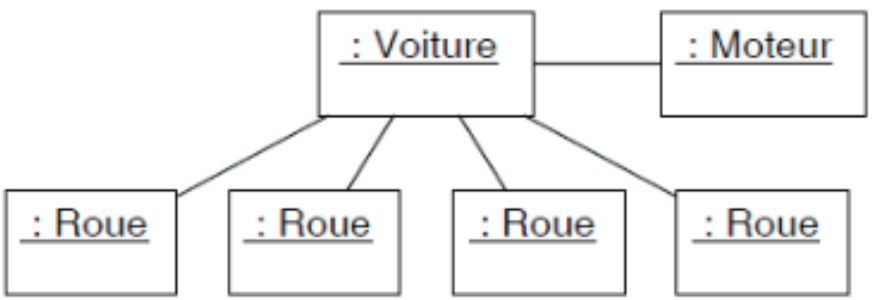
Définition d'un package

Définition

Un diagramme d'objets représente une vue statique d'un ensemble d'instance de classes.

Remarques

- Ils servent à donner donner des exemples et à valider le diagramme de classes.
- On parle de liens entre les objets et d'associations entre les classes.



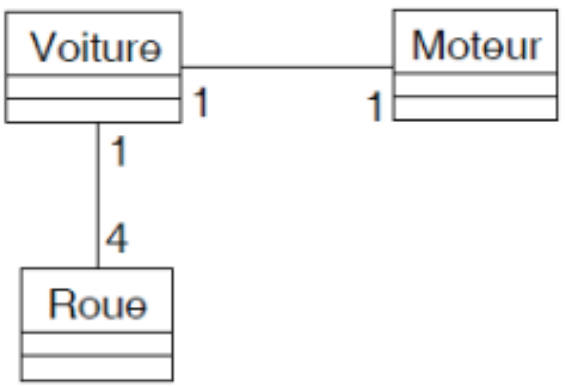


Diagramme de classes

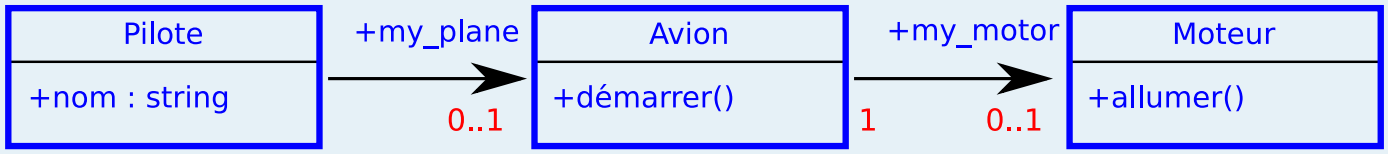


Diagramme de séquences

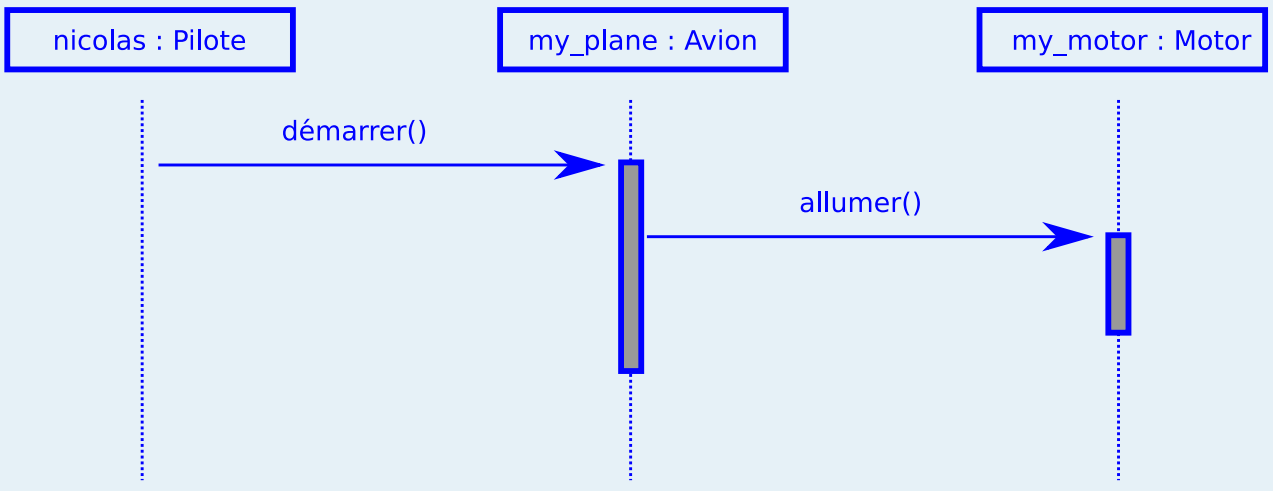


Diagramme de classes

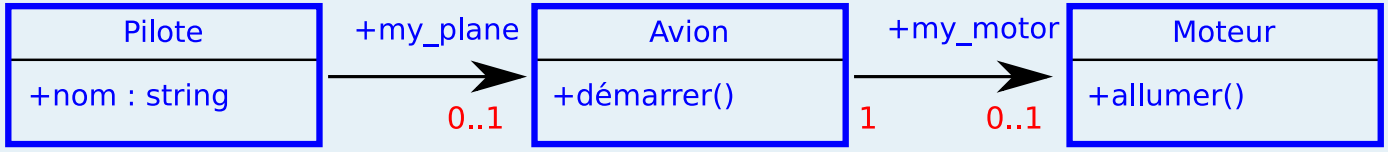


Diagramme de séquences

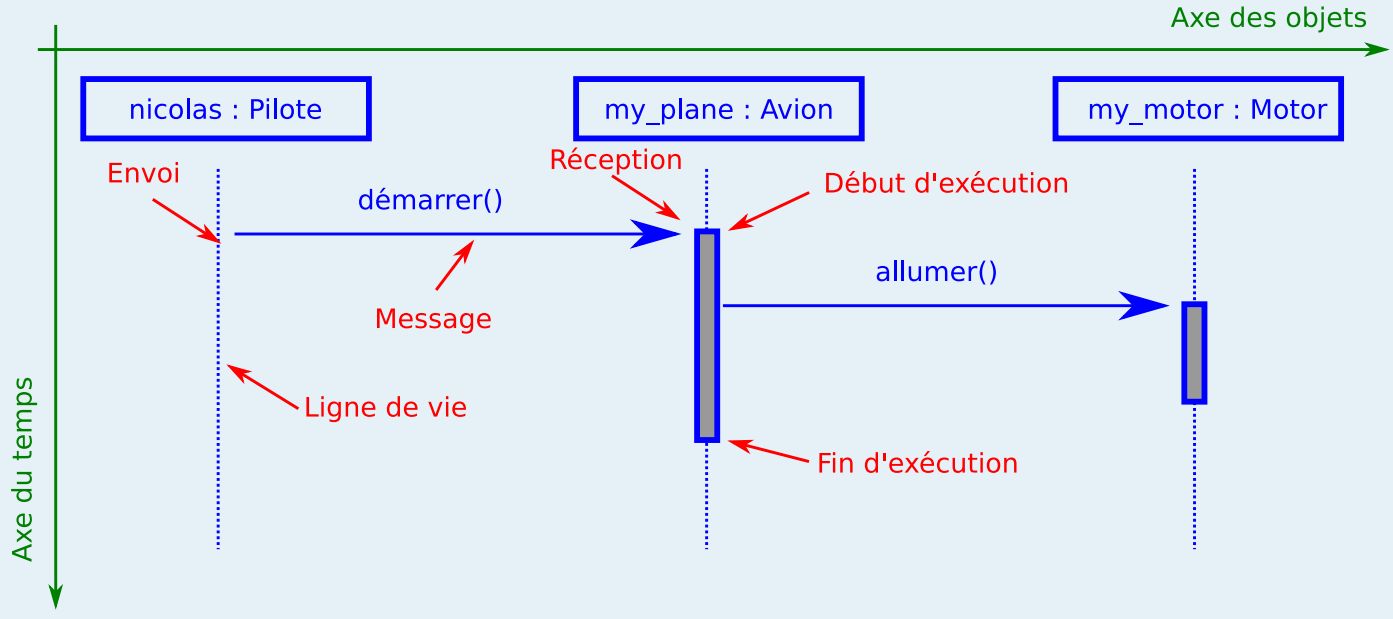
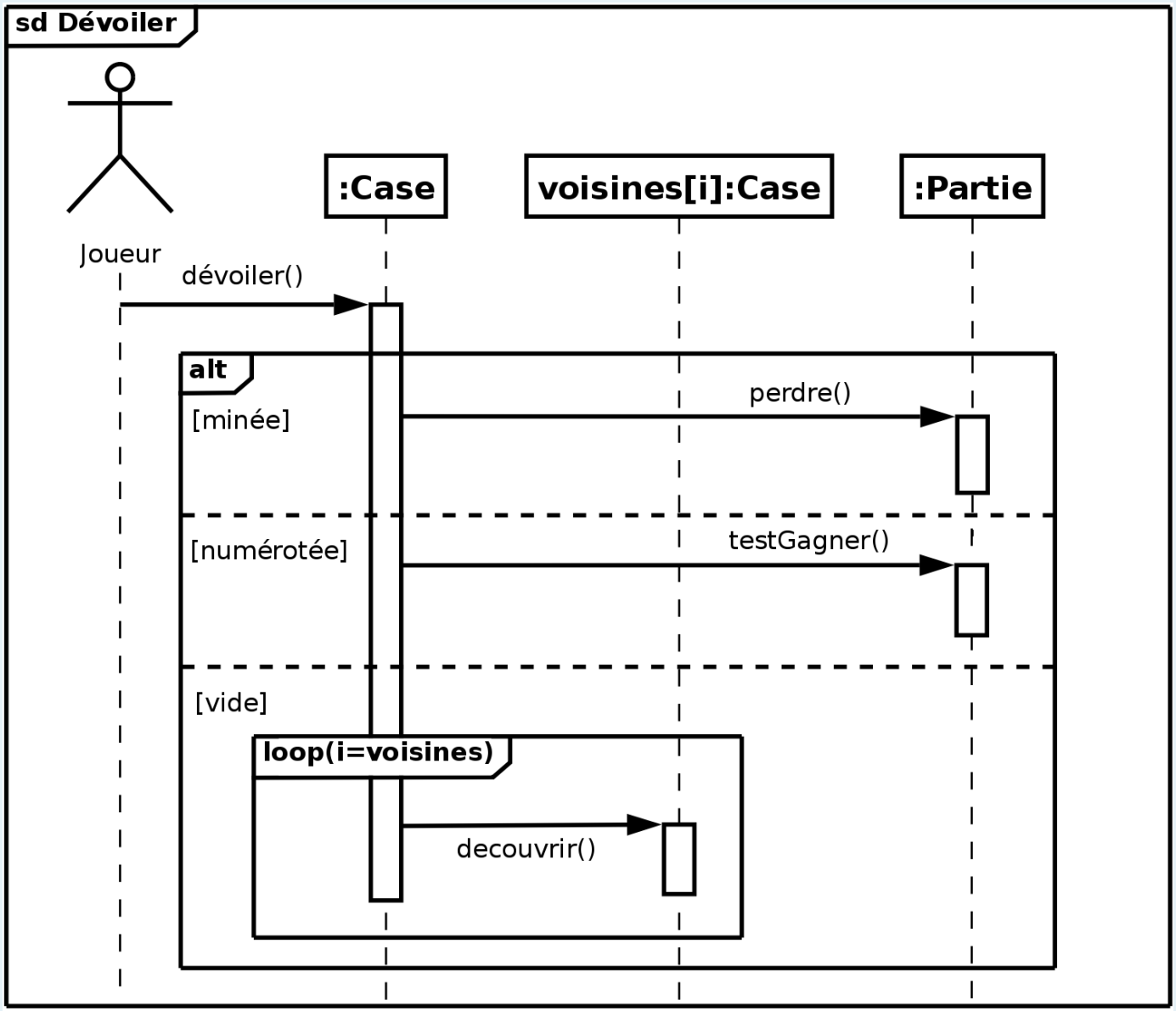


Diagramme de séquences



Conclusion

- *Diagramme de classes :*
  Classe, héritage, association (nom, cardinalité, rôle, navigabilité, agrégation, composition).
- *Diagramme d'objets :*
  Instance (objet), lien.
- *Diagramme de packages :*
  Dépendance,
- *Diagramme de séquences :*
  Ligne de vie, message (création, destruction, synchrone, asynchrone), fragment combinés (opt, alt, loop)