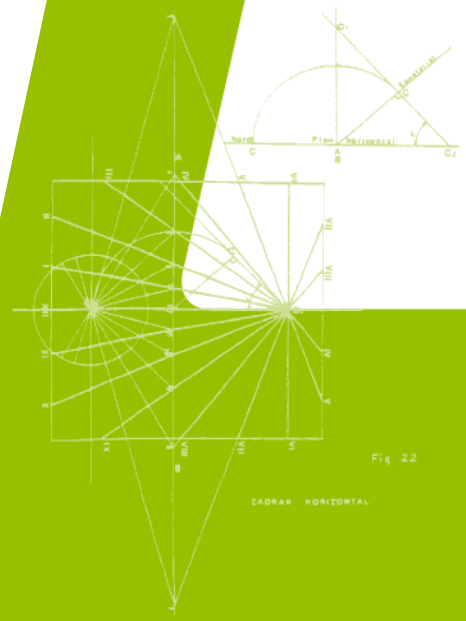
INF3055
2022-2023

Université de Yaoundé I

INF3055 : Conception Orientée Objet

Langage UML



Octobre 2022

Valéry MONTHE

valery.monthe@facsciences-uy1.cm

Bureau R114, Bloc pédagogique 1

UML
Unified Modelling Language

Introduction

- UML : langage de modélisation unifié.
- 1997: Unification de 3 modélisations objets : OMT, BOOCH, OOSE
- UML, c'est :
  - Une norme,
  - Un langage de modélisation objet,
  - Un support de communication.

Introduction

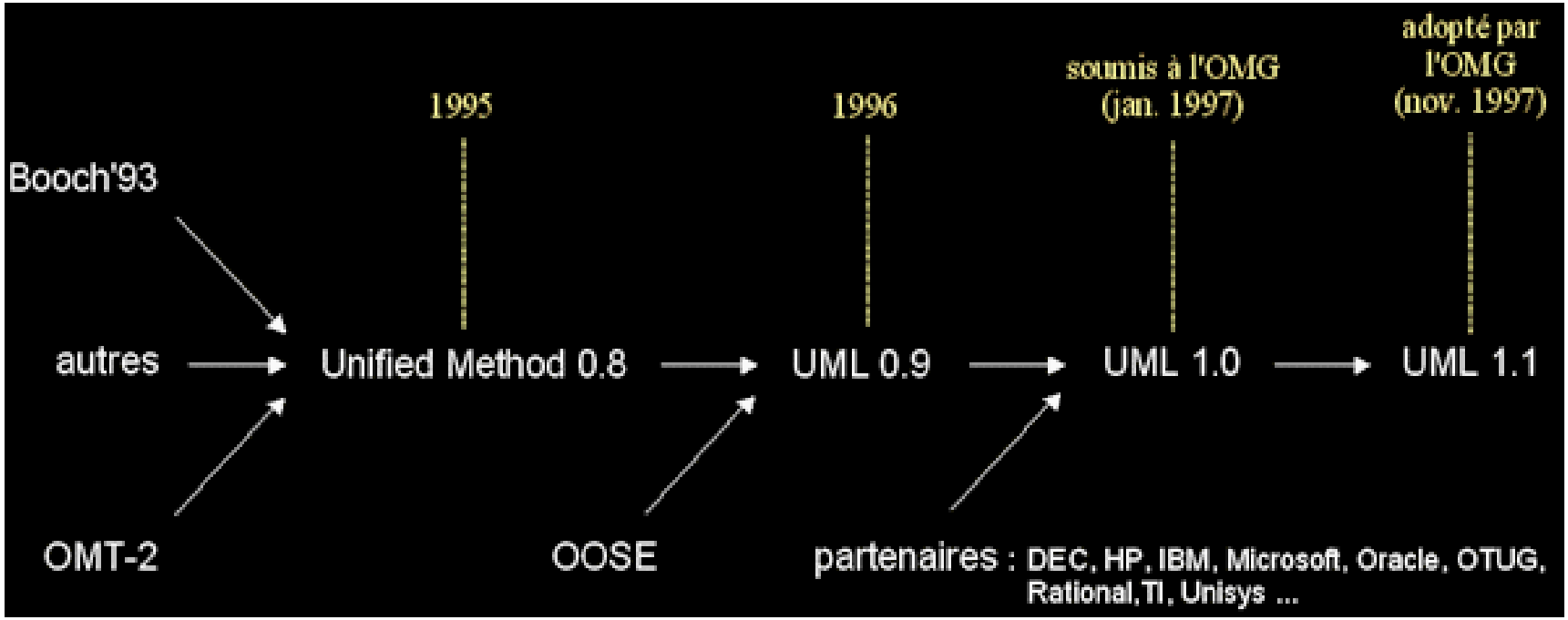
Méthode = langage(s) + démarche + outils

UML ne propose pas une démarche de modélisation.

=> **UML n'est pas une méthode**

Historique

Unification de 3 modélisations objets :
OMT, BOOCH, OOSE



Objectifs d'UML

- Montrer les limites d'un système et ses fonctions principales (utilisateur) : diagramme de C U
- Illustrer la réalisation des cas d'utilisation (ie les fonctions du systèmes) : diagrammes d'interaction, diagramme d'activités
- Modéliser la structure statique d'un système : diagramme de classes, d'objets, associations, contraintes
- Modéliser la dynamique, le comportement des objets : diagramme d'états transition
- Révéler l'implantation physique de l'architecture d'un système : diagramme de composants, diagramme de déploiement

Diagrammes d'UML

- **Vues statiques ou structurelles du système:**
  - Diagramme de cas d'utilisation
  - Diagramme de classes
  - Diagramme d'objets
  - Diagramme des composants
  - Diagramme de déploiement

- **Vues dynamiques ou comportementales du système :**
  - Diagramme d'etats-transition
  - Diagramme d'activites
  - Diagramme d'interaction
    - ✓ Diagramme de collaboration ou communication
    - ✓ Diagramme de séquences

Diagramme des cas d'utilisation

Objectifs

- Technique pour capturer les exigences fonctionnels d'un système.
  - ✓ Déterminer ses limites
  - ✓ Déterminer ce qu'il devra faire : point de vu utilisateur

- Pour cela
  - ✓ Déterminer les rôles qui interagissent avec le système : **acteur**
  - ✓ Déterminer les grandes catégories d'utilisation : **cas d'utilisation**
  - ✓ Décrire textuellement les interactions : scenario

Utilisation

- Passer du flou du cahier de charge à des fonctionnalités exprimées dans le langage du domaine.

- Un diagramme de cas d'utilisation
  - ✓ Ne représente pas le déroulement d'un système
  - ✓ Il dit ce qu'il fait (ou fera)
  - ✓ Sans dire comment il le fait (fera)

Acteur

- Idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec un système

- **Entité (humain ou machine) située hors du système**
  - ✓ permettant d'en déterminer les limites
  - ✓ jouant un rôle par rapport à lui
  - ✓ déclenchant une action entraînant une réaction du système (principal)
  - ✓ ou au contraire étant sollicité par le système au cours d'un Scénario (secondaire)

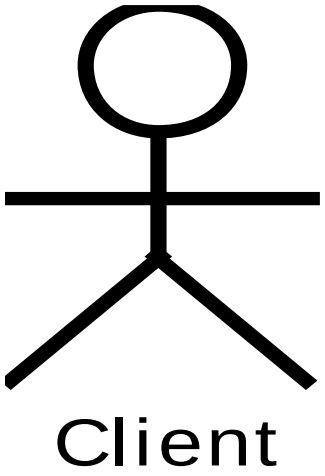
- **sont considérés comme acteurs d'un système :**
  - ✓ Des personnes qui l'utilisent (fonctions principales du système)
  - ✓ Matériel externe
  - ✓ Autres systèmes

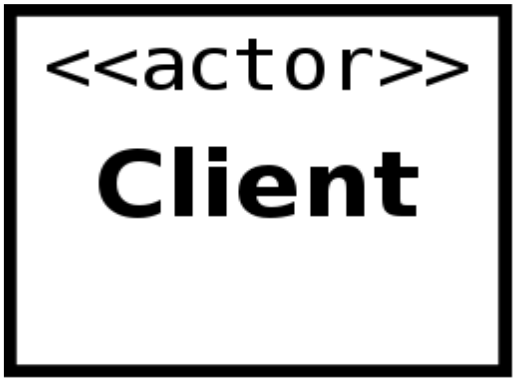
Acteur

- **Acteur principal**
  - ✓ Pour les cas d'utilisation qui rende service a cet acteur
  - ✓ Initie le cas d'utilisation par ses sollicitation
  - ✓ Obtient des résultats observables du système
  - ✓ Un seul par cas d'utilisation

- **Acteur secondaire**
  - ✓ Solliciter par le système pour des informations complémentaires
  - ✓ Les autres acteurs du cas d'utilisation

Acteur : représentation





Cas d'utilisation

- Unité cohérente représentant une fonctionnalité visible de l'extérieur
- Réalise un service de bout en bout, déclenchement, déroulement et fin, pour l'acteur qui l'initie
- Modélise un service rendu par le système
- Il est interne s'il n'est pas directement relie a un acteur

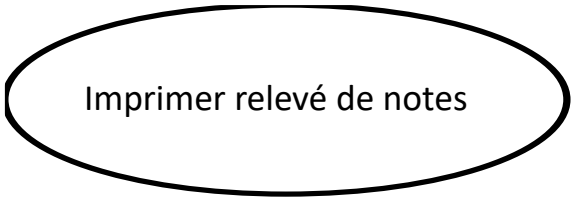
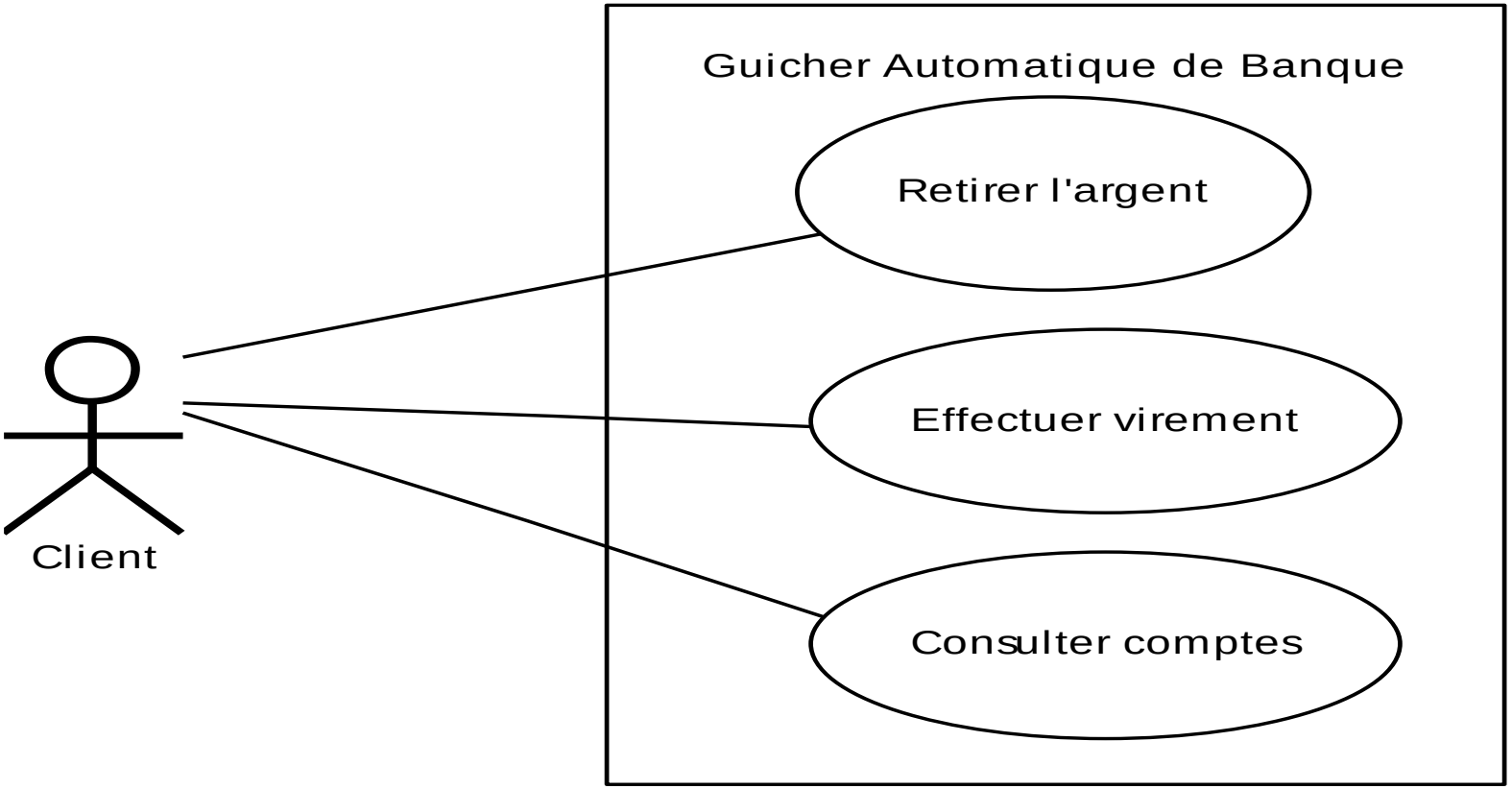


Diagramme des use case : Représentation

- Circonscrire le système par un rectangle et relier les acteurs a leur cas d'utilisation par des associations



Relations entre cas d'utilisation

- **Inclusion <<*include*>>**

  ✓ Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B.

  ✓ Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A.

- **Extension <<*extend*>>**

  ✓ A étend B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B.

  ✓ Exécuter B peut éventuellement entraîner l'exécution de A

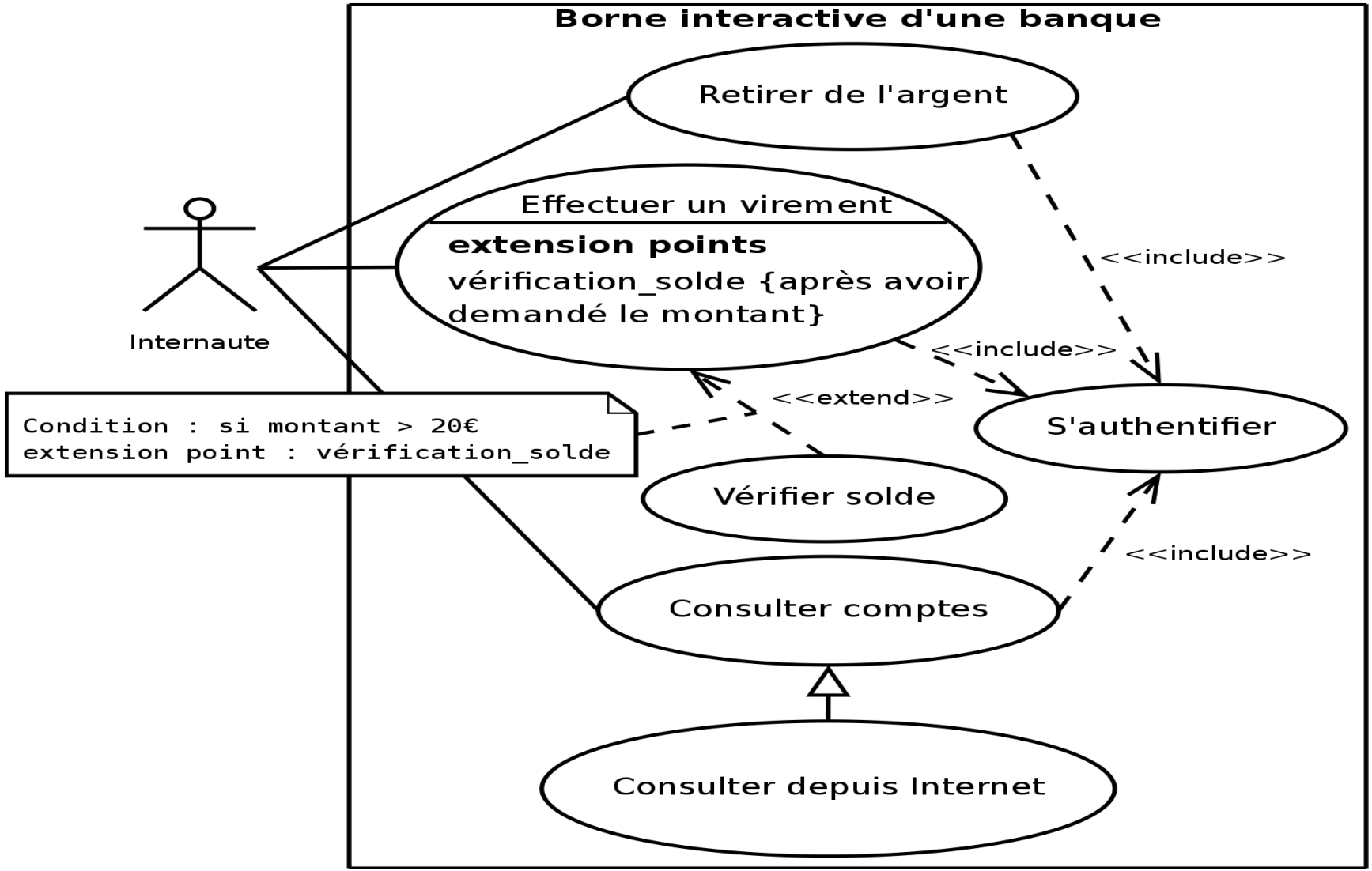
  ✓ Contrairement à l'inclusion, l'extension est optionnelle.

  ✓ Est souvent soumise à condition, exprimée sous la forme d'une note.

- **Généralisation**

  ✓ Un cas A est une généralisation d'un cas B si B est un cas particulier de A

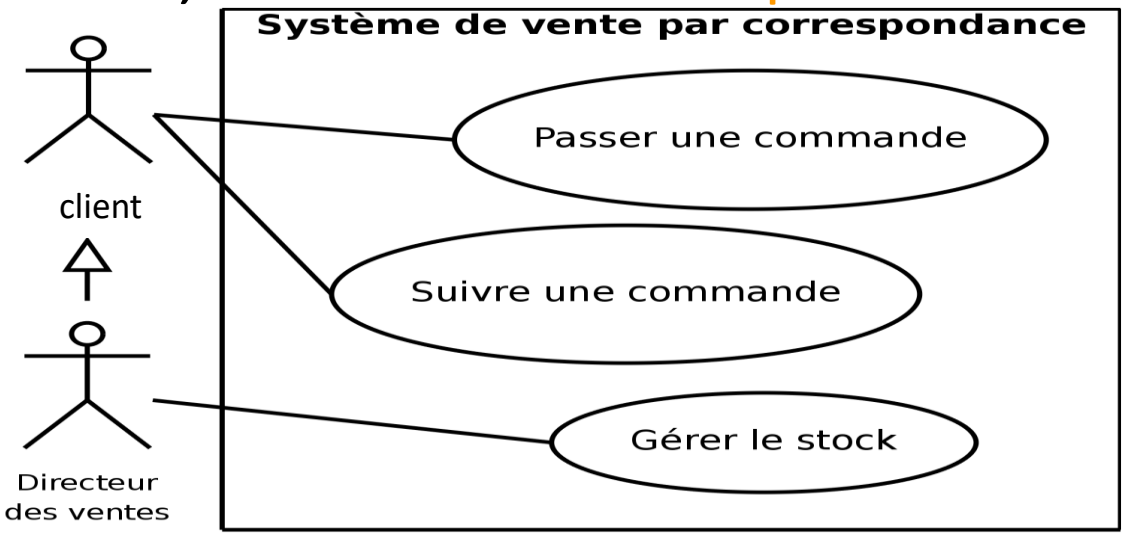
Relations entre cas d'utilisation



Relations entre acteurs

- la seule relation entre acteurs est la **généralisation**.

- Si un acteur A est une généralisation d'un acteur B
  - ✓ l'acteur A peut être substitué par l'acteur B.
  - ✓ tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.



Description textuelle des use case

- Explication du diagramme des cas d'utilisation, a travers une description textuelle des use case.

- Se présente en 2 parties :
- Identification du cas :
  - ✓**Nom :** à l'infinitif (ex : Réceptionner un colis).
  - ✓**Objectif : R**ésumée permettant de comprendre l'intention principale du cas d'utilisation.
  - ✓**Acteurs principaux :** Ceux qui vont réaliser le cas d'utilisation
  - ✓**Acteurs secondaires :** reçoivent des informations à l'issue de la réalisation du cas d'utilisation
  - ✓**Dates :** Les dates de créations et de mise à jour de la description courante.
  - ✓**Responsable :** Le nom des responsables.
  - ✓**Version :** Le numéro de version.
- Description du fonctionnement du cas:
  - ✓ Scenario nominale
  - ✓ Scenario alternatif
  - ✓ Scenario d'exception
  - ✓ Pré condition
  - ✓ Post condition

Description textuelle : Exemple 1/3

Pré-conditions :

- La caisse du DAB n'est pas vide.
- La connexion avec le système d'autorisation est opérationnelle.

Post-conditions :

- La caisse du DAB contient moins de billets qu'au début du cas d'utilisation (le nombre de billets manquants est fonction du montant du retrait).
- Une opération de retrait a été archivée (en cas de succès comme en cas d'échec).

Description textuelle : Exemple 2/3

DAB : scénario nominal

Cas d'utilisation : Retirer de l'argent au distributeur (DAB) avec une carte bancaire.

1. Le porteur de carte introduit sa carte dans le DAB.
2. Le DAB vérifie que la carte introduite est bien une carte bancaire.
3. Le DAB demande au porteur de carte de fournir son code d'identification.
4. Le porteur de carte entre son code d'identification.
5. Le DAB valide le code d'identification (par rapport à celui qui est codé sur la puce de la carte).
6. Le DAB demande une autorisation au système d'autorisation externe.
7. Le système d'autorisation externe donne son accord et indique le solde hebdomadaire.
8. Le DAB demande au porteur de carte de saisir le montant désiré du retrait.

Description textuelle : Exemple 3/3

DAB : alternatives

2a. La carte introduite n'est pas reconnue par le DAB.

1. Le DAB éjecte la carte et le cas d'utilisation se termine en échec.

5a. Le DAB détecte que le code saisi est erroné, pour la première ou deuxième fois.

1. Le DAB indique au porteur de carte que le code est erroné.

2. Le DAB enregistre l'échec sur la carte et le cas d'utilisation reprend à l'étape 5 du scénario nominal.

5b. Le DAB détecte que le code saisi est erroné, pour la troisième fois.

1. Le DAB indique au porteur de carte que le code est erroné pour la troisième fois.

2. Le DAB confisque la carte.

3. Le DAB informe le système d'autorisation externe et le cas d'utilisation se termine en échec.

Diagramme de classes

Objectifs

- Présenter la structure interne du système
- Fournir une représentation abstraite des objets qui vont interagir pour réaliser les cas d'utilisation.
- Vue statique car on ne tient pas compte du facteur temporel dans le comportement du système.
- modélise les concepts du domaine d'application
- Les principaux éléments de cette vue sont les classes et leurs relations

=> **Diagramme le plus utilise et le plus connu**

Classes

- Représentation d'un ensemble d'objets ayant les mêmes caractéristiques
  - ***Structure /comportement / relations / sémantique commune***
- Nom au singulier avec majuscule au début. Exemple : Fichier, Personne, Client
- Fournir une représentation abstraite des objets qui vont interagir pour réaliser les cas d'utilisation.
- modélise les concepts du domaine d'application
- Chaque élément particulier de cette famille est un objet (instance de la classe)

Caractéristiques d'une classe

- Permettent de spécifier son état et son comportement

  - ✓ Etat d'un objet : attributs et terminaison d'association = propriétés structurelles.

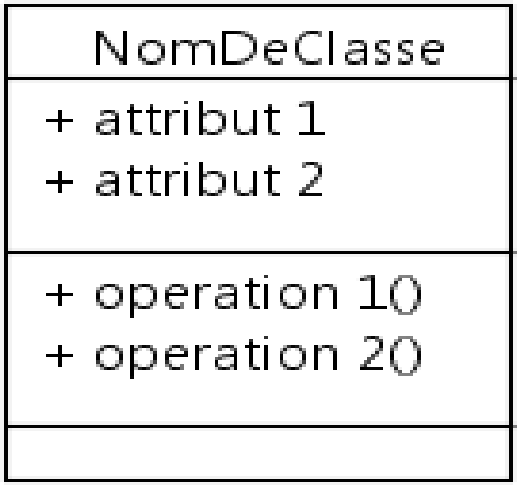
  - ✓ Comportement d'un objet : opération que l'on peut invoquer.

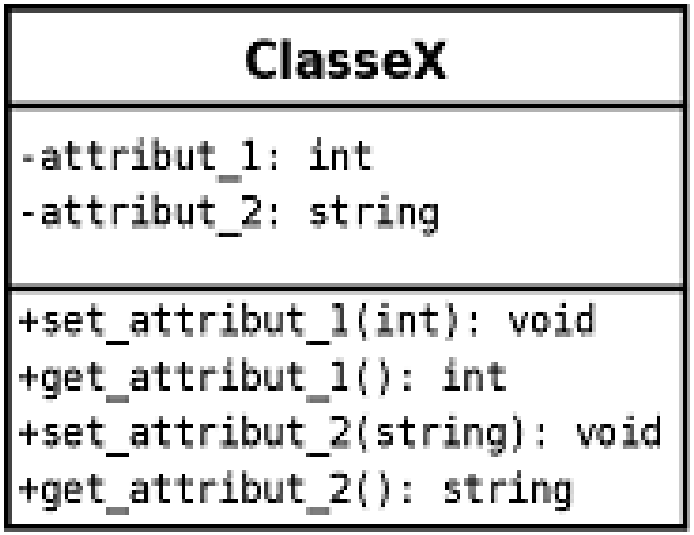
    - Opération = spécification (déclaration) d'une méthode.

Représentation d'une classe

- Un rectangle divisé généralement en trois compartiments:

  ✓ Le premier indique le nom de la classe
  ✓ Le deuxième ses attributs
  ✓ Le troisième ses opérations





Encapsulation, visibilité, interface

- **Encapsulation** : permet de définir les niveau de visibilité des éléments d'un conteneur.
- **Visibilité** : définie la possibilité pour un élément de modélisation de référencer un autre élément situe dans un autre conteneur.

✓ ***Public*** ou + : tout élément qui peut voir le conteneur peut également voir l'élément indiqué.

✓ ***Protected*** ou # : seul un élément situé dans le conteneur ou un de ses descendants peut voir l'élément indiqué.

✓ ***Private*** ou - : seul un élément situé dans le conteneur peut voir l'élément.

✓ ***Package*** ou ~ ou rien : seul un élément déclaré dans le même paquetage peut voir l'élément.

Les attributs

- **Attributs de classe**
  - ✓ Possède une valeur unique qui est partagée par toutes les instances de la classe.
  - ✓ Est déclaré static en java ou en C++
  - ✓ Les instances y ont accès mais ne possède pas une copie de la valeur
  - ✓ Pas besoin d'une instance de la classe pour y accéder .
  - ✓ Est toujours souligne

- **Attribut dérivé**
  - ✓ Calculé a partir d'autres attributs et des formule
  - ✓ Sont symbolise par l'ajout d'un « /» devant leur nom

Les méthodes

- **Méthode de la classe**
  - ✓ Dans une classe une opération (même nom et même type de paramètre) doit être unique.
  - ✓ Opération surchargée : le nom apparait plusieurs fois avec des paramètres différents.
  - ✓ Deux Operations ne peuvent pas se distinguée que par le type de valeur retournée.
- **Méthode de classe**
  - ✓ N'a pas accès aux attributs de la classes (des instances de classe)
  - ✓ Ne peut manipuler que des attributs de classe et ses propres paramètres
  - ✓ Ne nécessite pas l'existence d'une instance de classe pour y accéder
  - ✓ Est soulignée
- **Déclaration**
  - ✓ **<visibilité> <nom_méthode> ([<paramètre_1>, ... , <paramètre_N>]) : [<type_renvoyé>] [{<propriétés>}]**

Méthodes et classes attraites

- **Méthode abstraite**
  - ✓ On connait son en tête, mais pas la manière dont elle est réalisée

- **Classe abstraite**
  - ✓ Définie au moins une méthode abstraite
  - ✓ Classe qui hérite d'une classe contenant une méthode abstraite non encore réalisée.
  - ✓ Ne peut être instanciée, est vouée a se spécialiser
  - ✓ Peut contenir des méthodes concrètes

- **Interface**
  - ✓ Classe abstraite pure, i.e qui ne comporte que des méthodes abstraites.

Relations entre classes

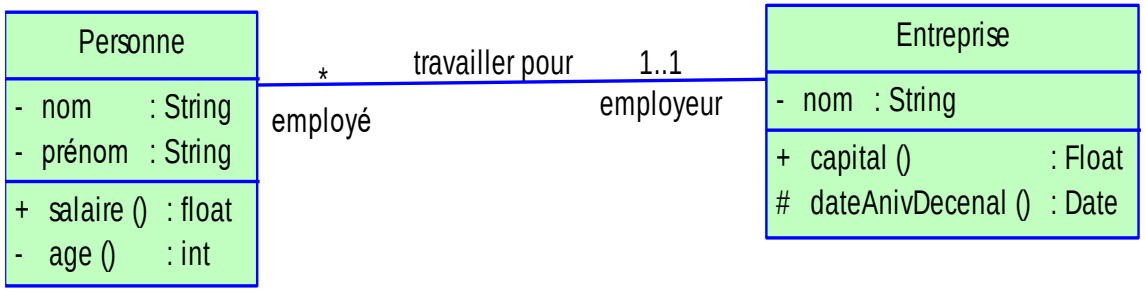
- Une association est une relation entre 2 classes ou plus qui décrit une connexion structurelle entre leurs instances.

- **Association binaire**
  - ✓ Matérialise par un trait plein entre les 2 classes.
  - ✓ Peut porter un nom
  - ✓ Peut avoir un sens de lecture
  - ✓ Réflexive, si ses 2 extrémités pointent vers la même classe.

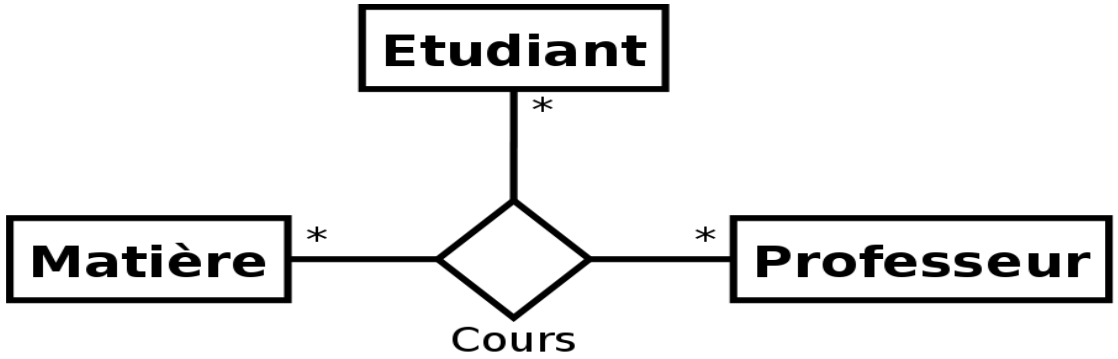
- **Association n-aire (n>2)**
  - ✓ Représentée par un losange blanc, avec des liens allant vers chaque classe.
  - ✓ Si elle a un nom, on le met a proximité du losange.

Relations entre classes

Association binaire

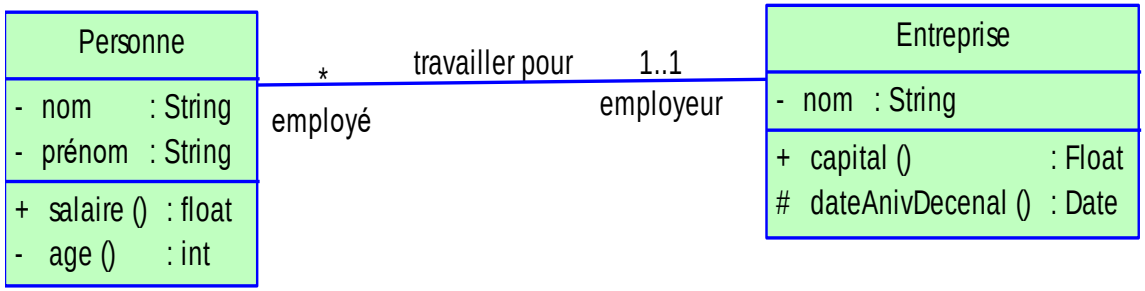


Association ternaire



Association : nommage de rôles

- Un rôle représente une extrémité d'une association



Multiplicité ou cardinalité

- Définie le nombre d'objets de la classe associée, qui peuvent participer a la relation.

- Exemple :
  - ✓ Exactement un : 1 ou 1:1
  - ✓ Au plus un : 0..1
  - ✓ Au moins 1 : 1..*
  - ✓ De 1 a 6 : 1..6
  - ✓ Plusieurs : * ou 0..*

- **Association binaire**
  - ✓ Matérialise par un trait plein entre les 2 classes.
  - ✓ Peut porter un nom
  - ✓ Peut avoir un sens de lecture
  - ✓ Réflexive, si ses 2 extrémités pointent vers la même classe.

- **Association n-aire (n>2)**
  - ✓ Représentée par un losange blanc, avec des liens allant vers chaque classe.
  - ✓ Si elle a un nom, on le met a proximité du losange.

Multiplicité ou cardinalité

- Remarques :

Pour une association n-aire, la multiplicité minimale doit en principe, mais pas nécessairement être de 0.

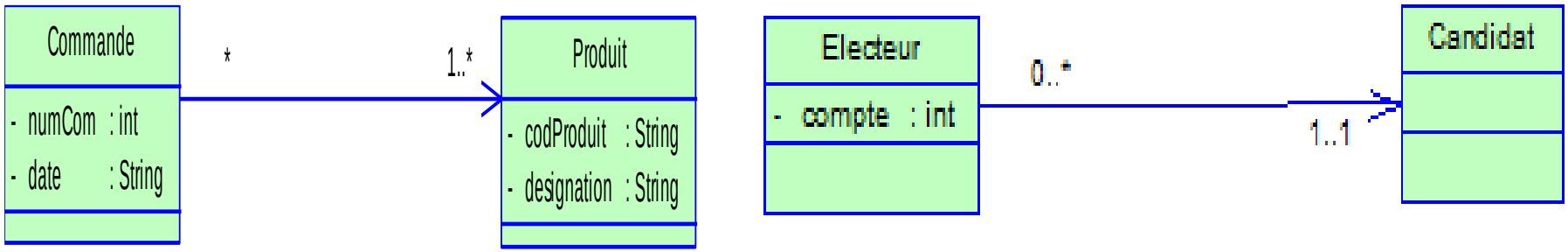
En effet, une multiplicité minimale de 1 (ou plus) sur une extrémité implique qu'il doit exister un lien (ou plus) pour TOUTES les combinaisons possibles des instances des classes situées aux autres extrémités de l'association n-aire.

Association : navigabilité

- La navigabilité indique s'il est possible de traverser une association.
- Graphiquement, on représente la navigabilité par une flèche du cote de la terminaison navigable.
- Par défaut une association est navigable dans les deux sens.

Exemple :

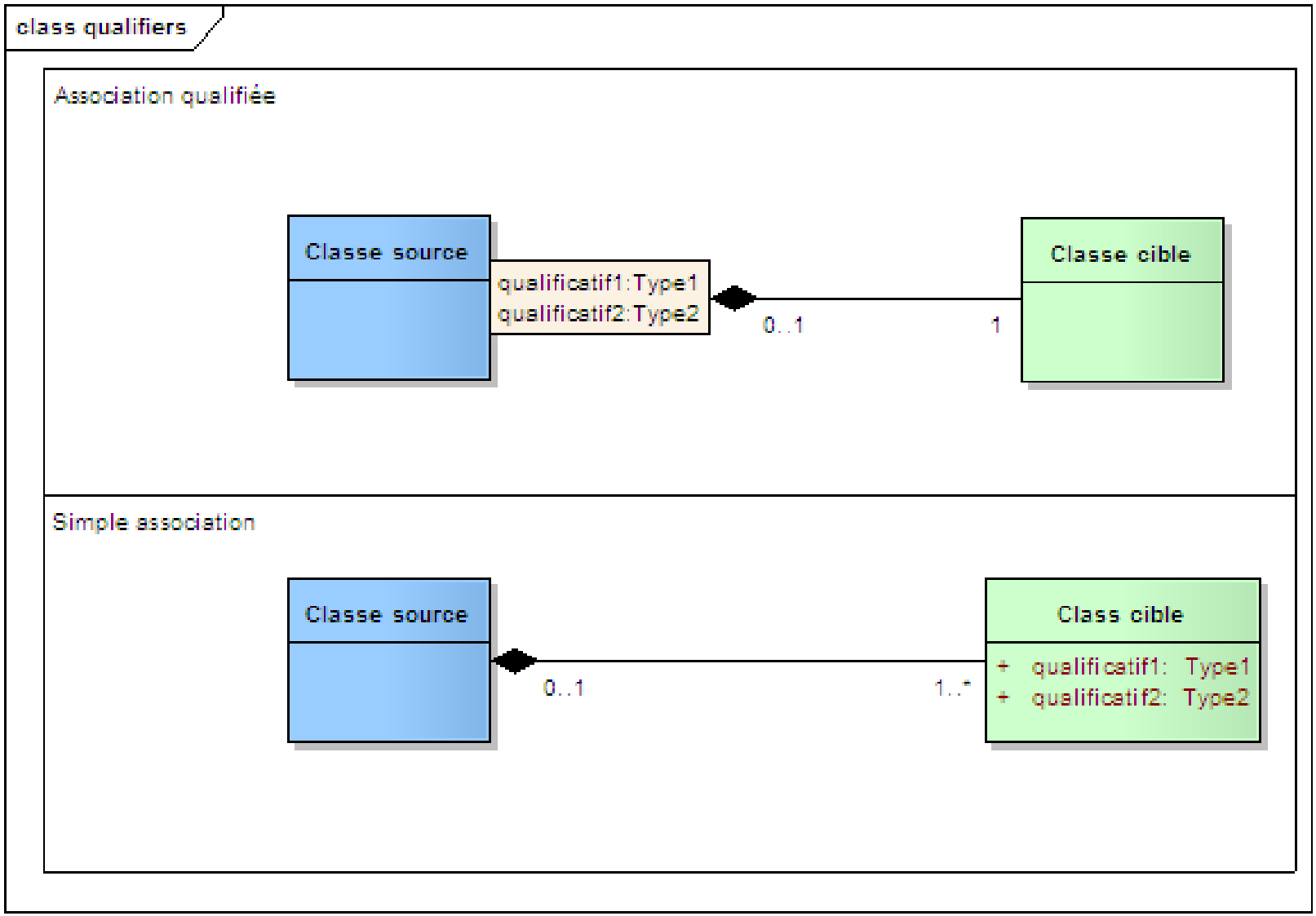
1. Une commande connait ses produit, mais le produit ne connait pas la commande.
2. Un électeur connait les candidats, mais un candidat ne connait pas les électeurs. On ne peut donc pas naviguer du candidat vers l' électeur.



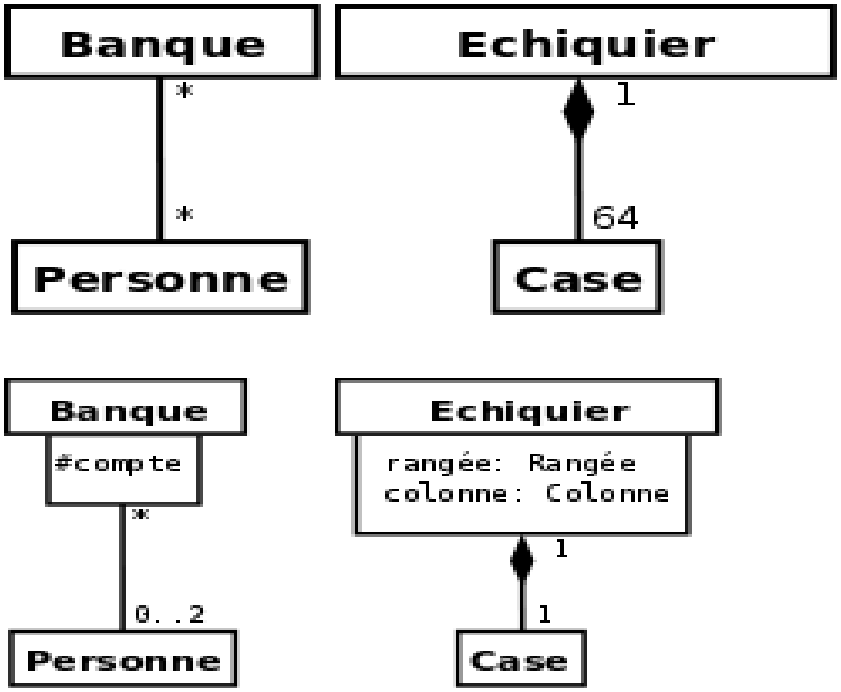
Qualification

- Permet de restreindre la portée d'une association a quelques éléments (1 ou plusieurs attributs) cibles de la classe.
  - ✓ Les attributs permettant de cibler le sous-ensembles d'objets sont appelés **qualificatifs**
  - ✓ L'objet sélectionné par la valeur du qualificatif est appelé **objet cible**.
  - ✓ L'association est appelée **association qualifiée**.
  - ✓ La classe qui porte le qualificatif est appelée **classe qualifiée** et ses objets, **objets qualifiés**.
- Un qualificatif agit toujours sur une association dont la multiplicité est plusieurs du cote cible.
- Un objet qualifié et une valeur du qualificatif génère un objet cible unique

Qualification



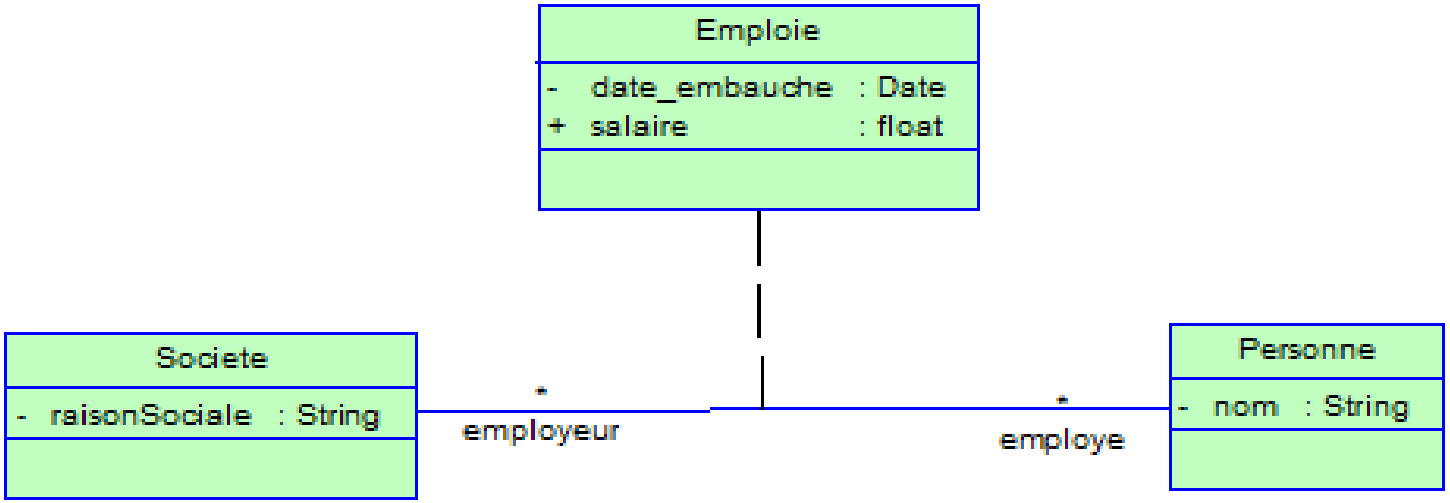
Qualification : exemple



- Une instance du couple **{Banque, compte}** est en association avec zéro à deux instances de la classe **Personne**.
- Une instance de la classe Personne peut être associée à plusieurs (zéro compris) instances du couple {Banque, compte}.
- Dans tous les cas, une instance du couple **{Personne, compte**} est en association **avec une instance unique de la classe Banque**.

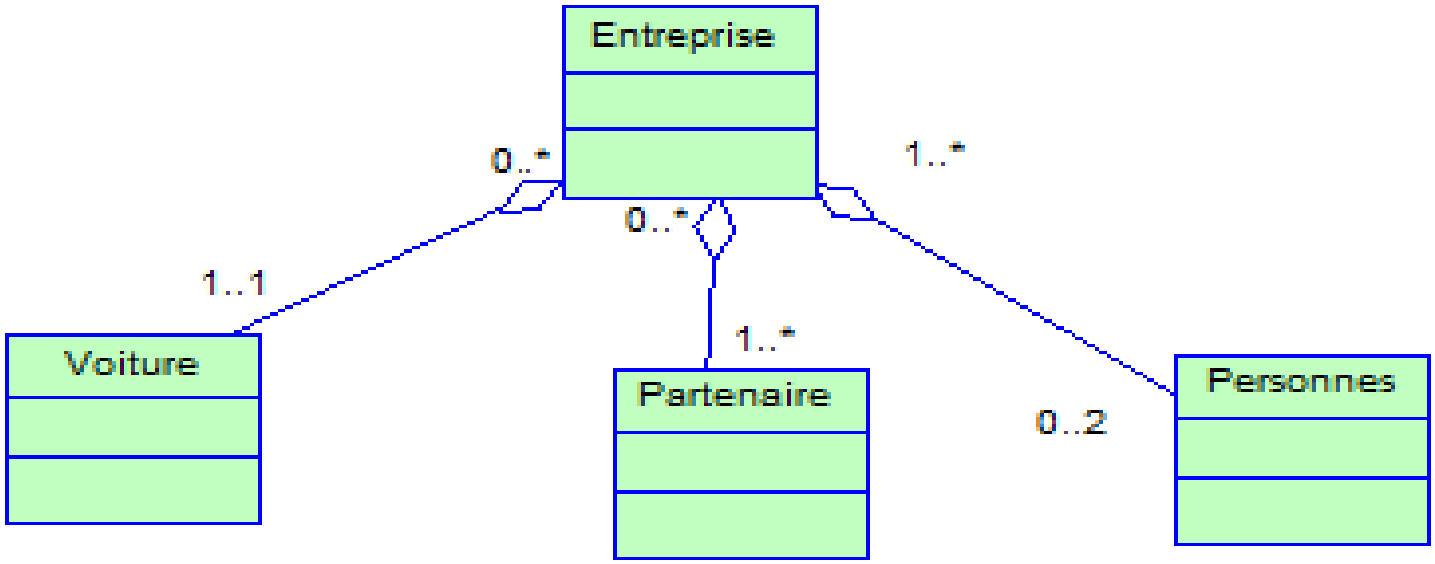
Classe d'association

- Association possédant les caractéristiques d'une association et celle d'une classe.
- Est utilise quand une association doit posséder des propriétés
- NB : il n'est pas possible de relier une classe d'association a plus d'une association.



Agrégation

Association représentant une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. On a une classe plus importante (tout) que l'autre (partie).

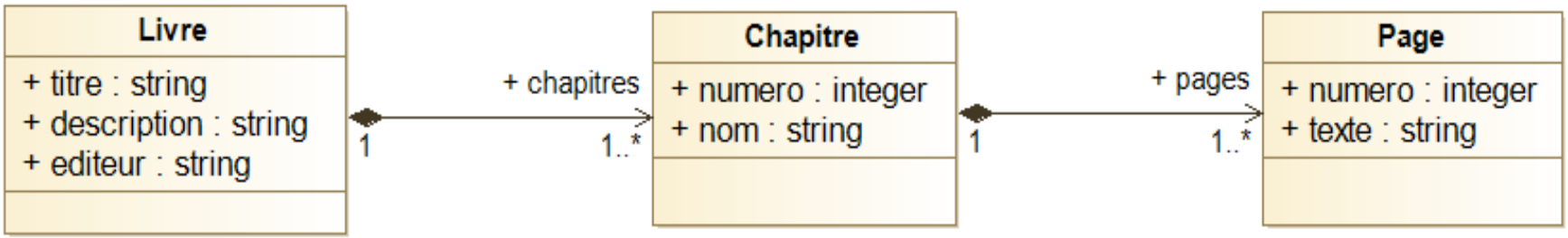


Composition

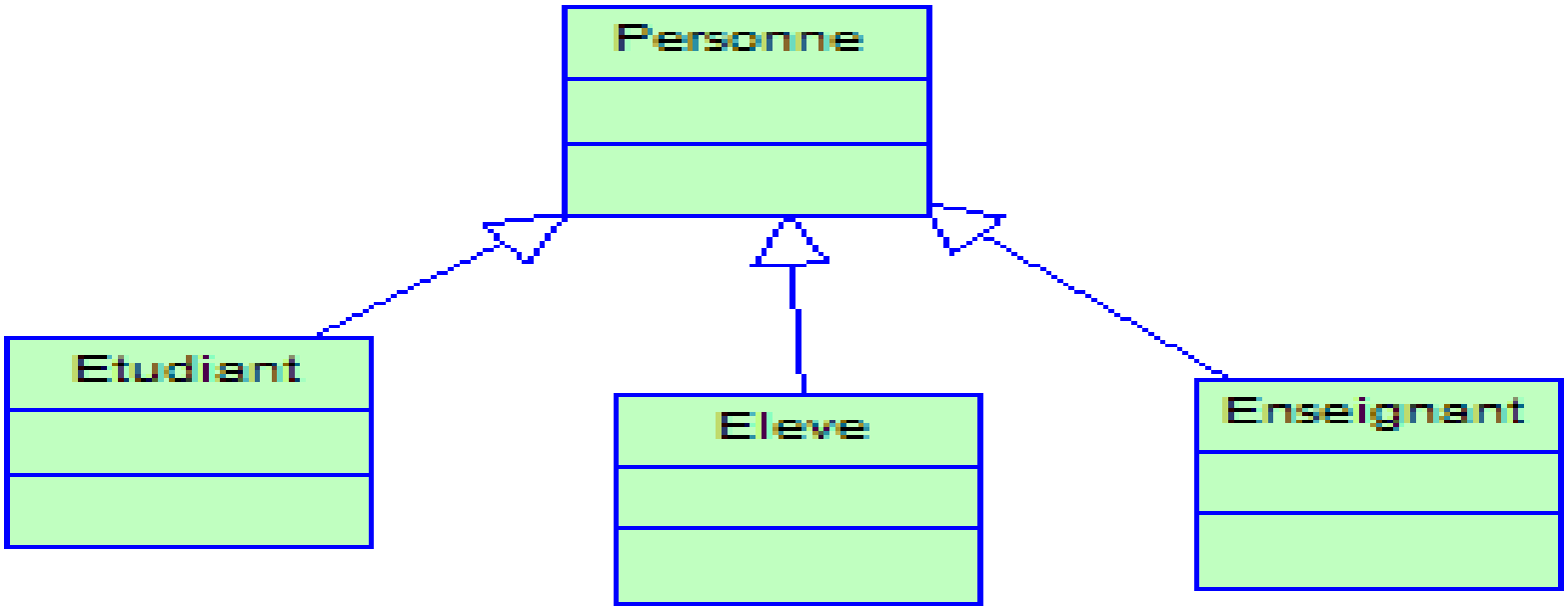
• Egalement appelée agrégation composite, décrit une contenance structurelle entre instances.

• La destruction de l'objet composite implique la destruction de ses composants.

• Une instance de la partie appartient toujours à au plus une instance de l'élément composite : la multiplicité du côté composite ne doit pas être supérieure à 1 (i.e. 1 ou 0..1).



Généralisation et héritage



Dépendance

•Relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle.

•Est représentée par un trait discontinu orienté.

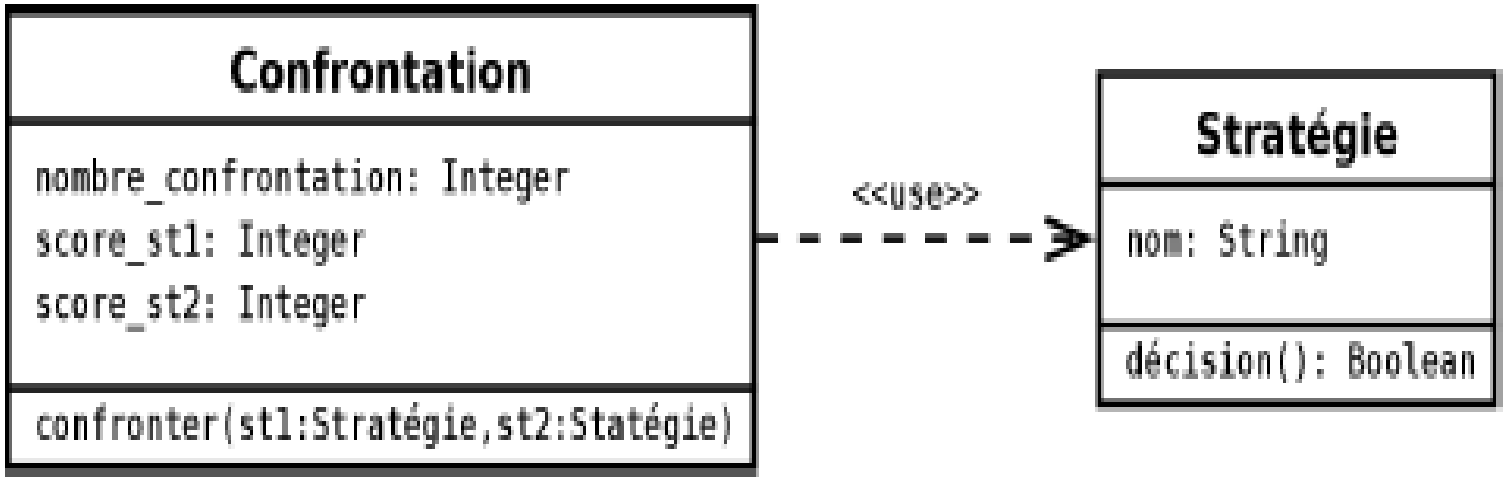
•La modification de la cible peut impliquer une modification de la source.

•La dépendance est souvent stéréotypée.

•Est utilisée souvent quand une classe utilise une autre comme argument dans la signature d'une opération.

Dépendance : Exemple

Exemple: la classe Confrontation utilise la classe Stratégie car la classe Confrontation possède une méthode ***confronter*** dont deux paramètre sont du type Stratégie.



Interface

C’est un classeur, stéréotypé « *interface* »

A pour rôle de regrouper un ensemble de propriétés et d’opérations assurant un service cohérent.

Est représentée comme une classe (stéréotype « interface»).

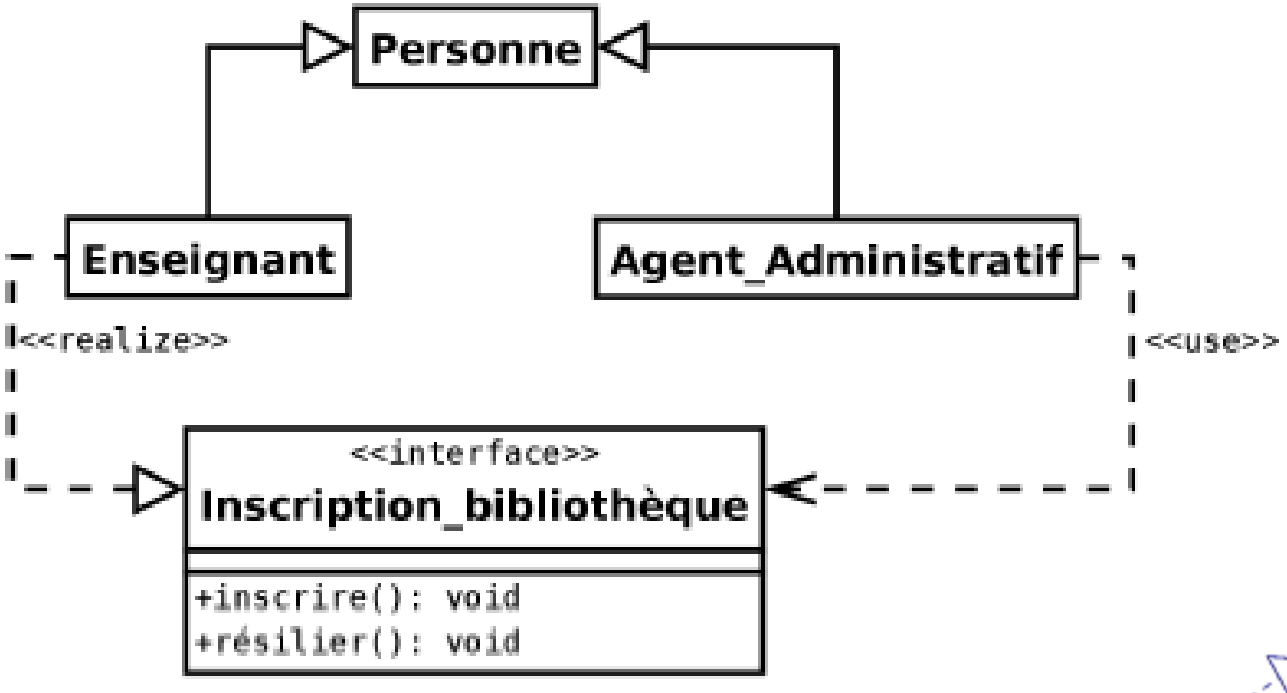
Elle **doit être réalisée par au moins une classe** et peut l’être par plusieurs.

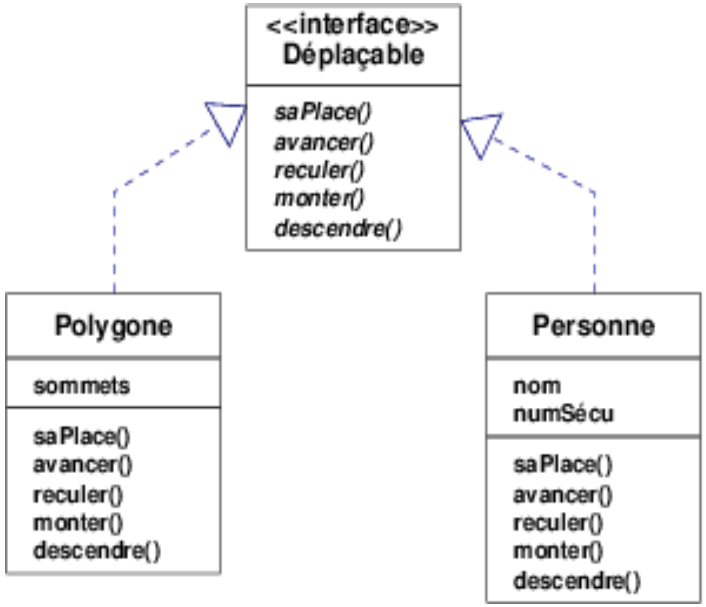
Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « *realize* ».

Une classe peut très bien **réaliser plusieurs interfaces.**

Une classe (classe cliente de l’interface) peut dépendre d’une interface (interface requise). On représente cela par une relation de dépendance et le stéréotype « *use*

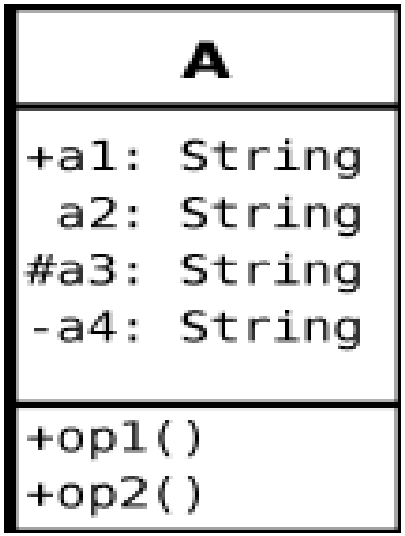
Interface : Exemple





Implémentation

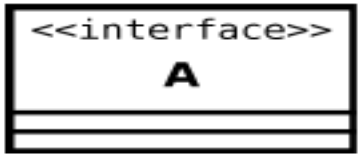
- Une classe avec attributs et opérations




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

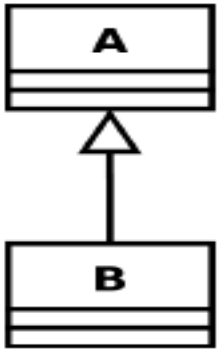
Implémentation

Interface




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

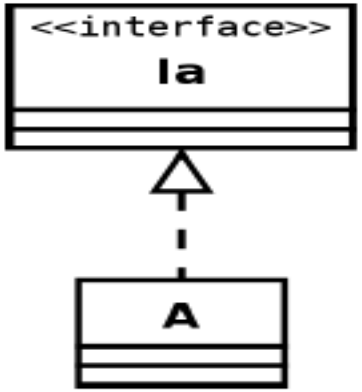
Héritage simple




```
public class A {
   ...
}

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

Réalisation d'une interface par une classe

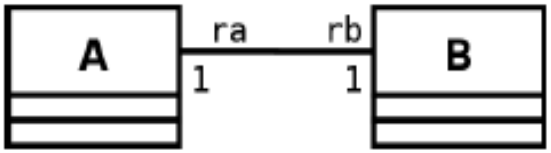



```
public interface Ia {
   ...
}

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

Implémentation

Association bidirectionnelle 1 vers 1



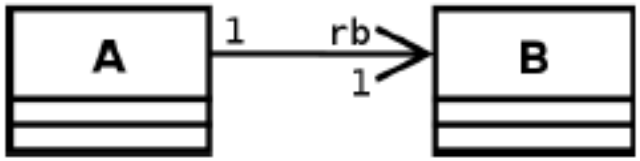

```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ){
      if ( b.getA() != null ) {   // si b est déjà connecté à un autre A
        b.getA().setB(null);    // cet autre A doit se déconnecter
      }
      this.setB( b );
      b.setA( this );
    }
  }
  public B getB()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

Implémentation

```
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

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
}
```

Implémentation

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);
    }
  }
```

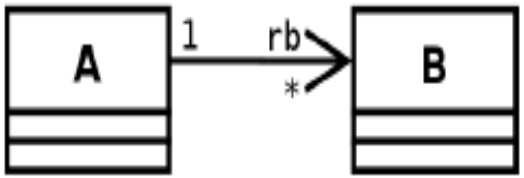
Implémentation

```
}

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

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
}
```

Diagramme d’objets

Objectifs

- Représente des objets (instances de classes) et leurs liens (instances de relations).

- Donner une vue figée de l'état d'un système à un instant donné.

- 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.

Un diagramme d'objets ne montre pas l'évolution du système dans le temps.

Diagramme d'Objets : Exemple 1

- Le diagramme de classes ci dessous montre qu'une entreprise emploie au moins deux personnes et qu'une personne travaille dans au plus deux entreprises.

- Le diagramme d'objets modélise lui une entreprise particulière qui emploie trois personnes.

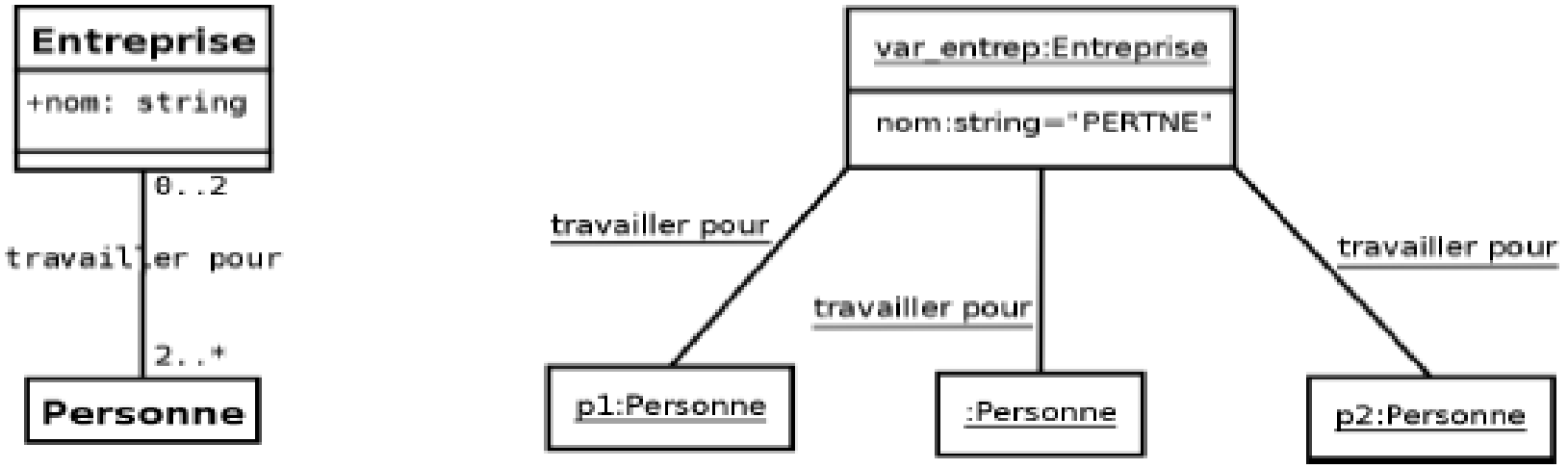
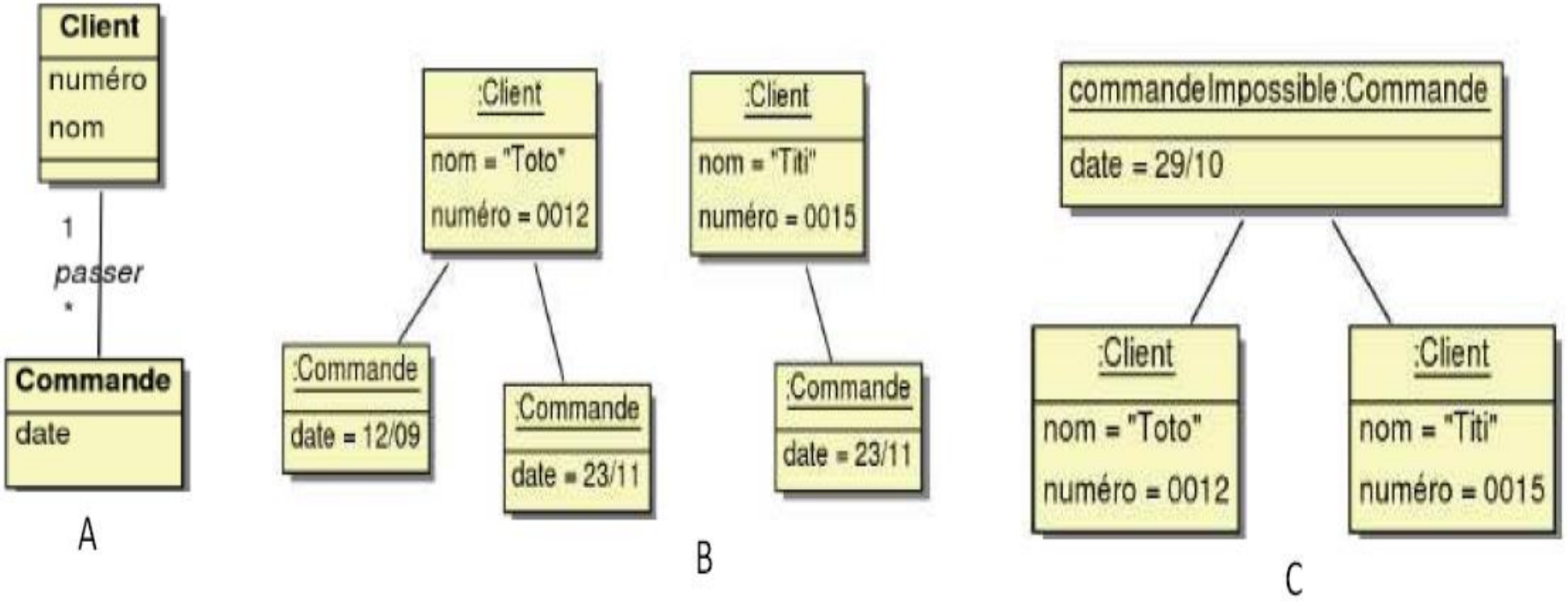
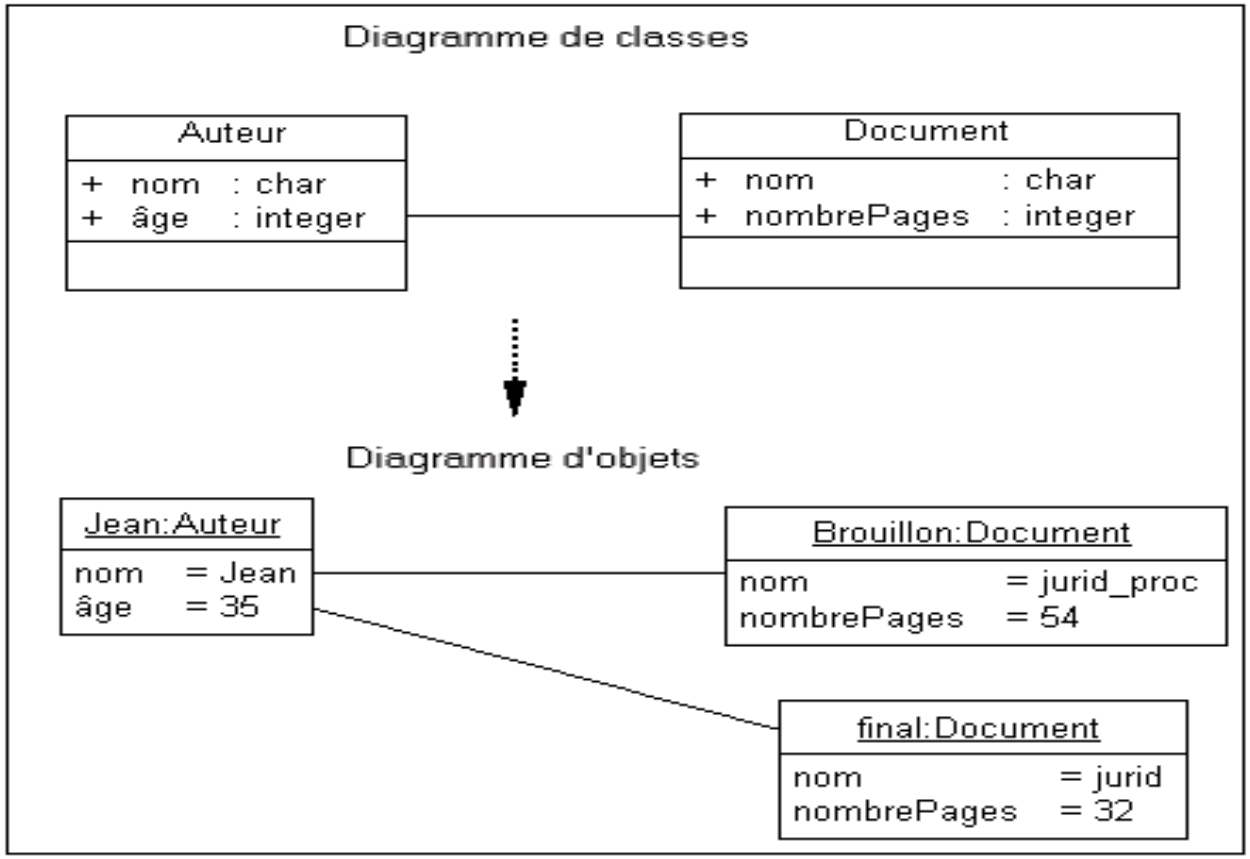


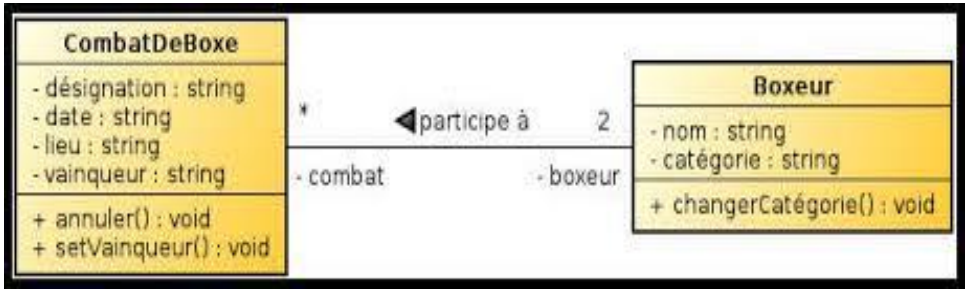
Diagramme d'Objets : Exemple 2



Les diagrammes d'objets sont – ils cohérents avec le diagramme de classes?

Diagramme d'Objets : Exemple 3





Relation de dépendance d'instanciation

- Elle est stéréotypée « instanceof » et décrit la relation entre un classeur et ses instances. Elle relie, en particulier, les liens aux associations et les objets aux classes.

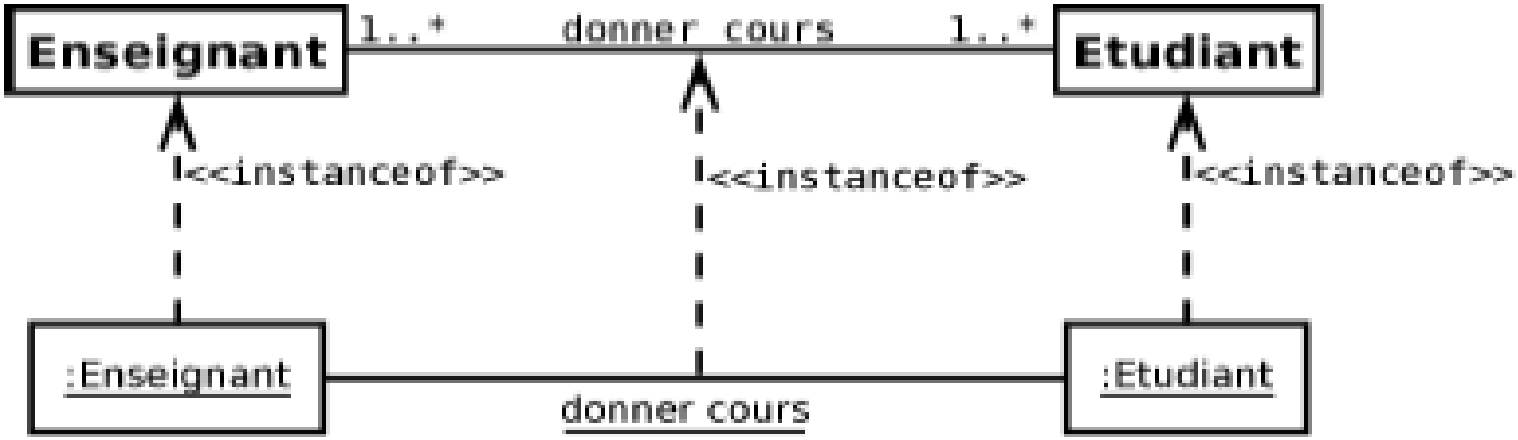


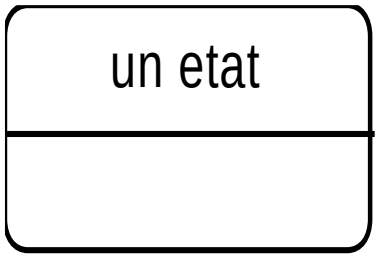
Diagramme d’états-transitions

Objectifs

- Ils décrivent le comportement interne d'un objet a l'aide d'un automate à états finis.
- Présentent les séquences possible d'états et d'action qu'une instance de classe peut traiter au cours de son cycle de vie en réaction a des événements.
- Spécifient aussi parfois des comportements internes des cas d'utilisation, des sous système.
- Rassemblent et organisent les états et les transitions d'une classe donnée.
- Il est souhaitable de réaliser un diagramme d'état transition pour chaque classe possédant un comportement dynamique important

Etat

- Période dans la vie d'un objet pendant laquelle ce dernier attend un événement ou accomplit une activité.
- Un objet peut passer par une série d'états pendant sa durée de vie.
- On le représente par un rectangle aux coins arrondis.



Etat initial et final

- Ce sont des pseudos états.
- Le premier indique l'état de départ, par défaut, lorsque le diagramme d'états-transitions est invoqué. Lorsqu'un objet est créé, il entre dans l'état initial.
- Le second indique que le diagramme d'états-transitions, ou l'état enveloppant, est terminé.

état initial

état final

Événement

- C' est quelque chose qui se produit pendant l'exécution d'un système et qui doit être modélisé.
- Il se produit à un instant précis et est dépourvu de durée.
- Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.
- Plusieurs types :

  signal, appel, changement et temporel.

Eléments d'un diagramme d'états-transitions

Événement de type signal (signal) : véhicule une communication asynchrone.

Syntaxe : **<nom_événement> ([<paramettre> : <type>] [; <paramettre> : <type>]; …)**

Événement d'appel (call) : réception de l'appel d'une opération par un objet.

- Les paramètres de l'opération sont ceux de l'événement d'appel.
- La syntaxe d'un événement d'appel est la même que celle d'un signal.
- Ce sont des méthodes déclarées au niveau du diagramme de classes.

Événement de changement (change) : généré par la satisfaction (passage de faux à vrai) d'une expression booléenne sur des valeurs d'attributs.

- Manière déclarative d'attendre qu'une condition soit satisfaite.
- Syntaxe : **when (<condition_booléenne>)**

Événement temporel (after ou when) : généré par le passage du temps.

- Sont spécifiés soit de manière absolue (date précise), soit de manière relative (temps écoulé).
- Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant.
- La syntaxe de manière relative est : **after (<durée>)**

Transition

- Définit la réponse d'un objet à l'occurrence d'un événement.

- Elle lie, généralement, deux états E1 et E2 et indique qu'un objet dans un état E1 peut entrer dans l'état E2 et exécuter certaines activités, si un événement déclencheur se produit et que la condition de garde est vérifiée.

- Syntaxe : **[<événement>] ['[' <garde> ']'] ['/' <activité>]**

- Le même événement peut être le déclencheur de plusieurs transitions quittant un même état.

- Chaque transition avec le même événement doit avoir une condition de garde différente.

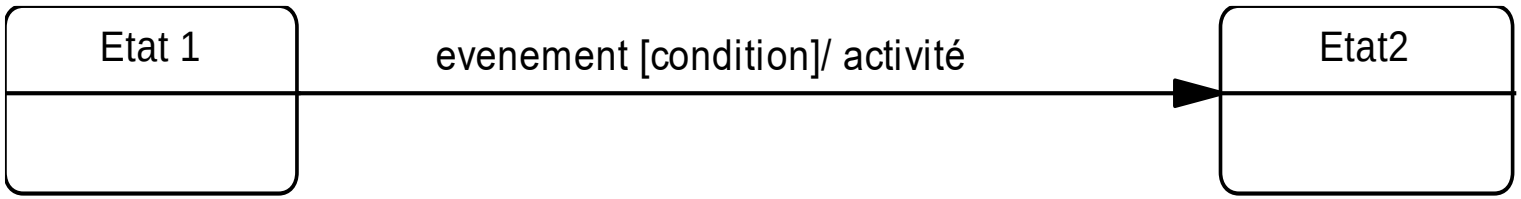
Effet d'une transition

- spécifié par **'/' <activité>** dans la syntaxe
- Il s'agit généralement d'une activité qui peut être :

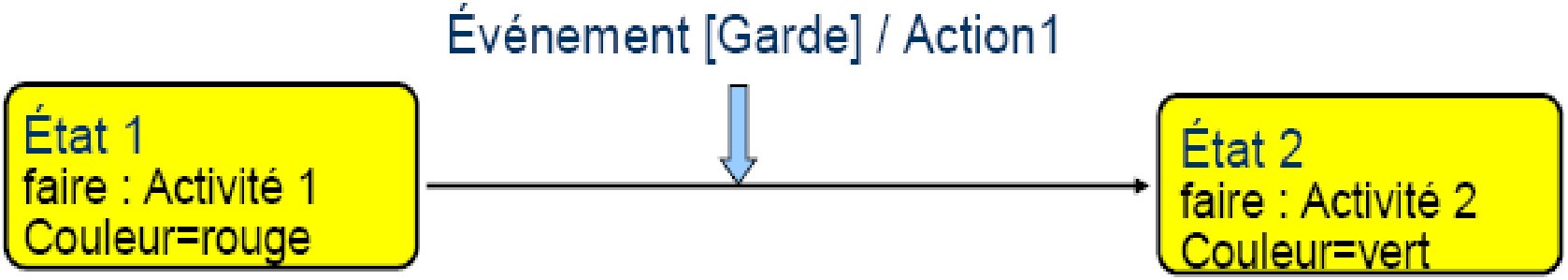
  - ✓ une opération primitive comme une instruction d'assignation ;
  - ✓ l'envoi d'un signal ;
  - ✓ l'appel d'une opération ;
  - ✓ une liste d'activités, etc.

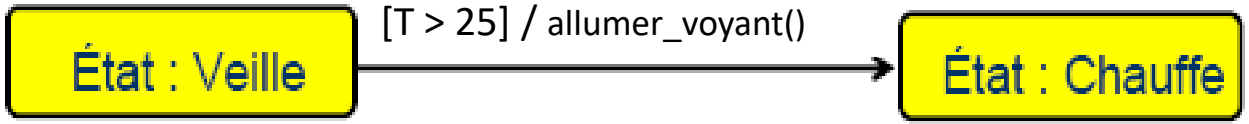
Transition externe

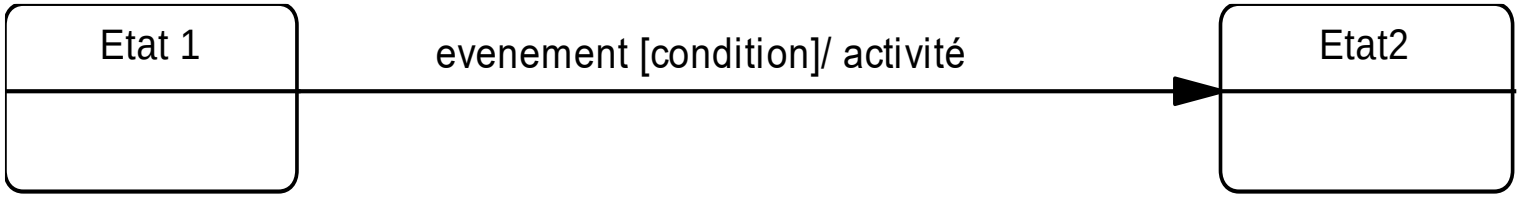
- Transition qui modifie l'état actif.
- Type de transition le plus répandu.
- représentée par une flèche allant de l'état source vers l'état cible.



Transition externe



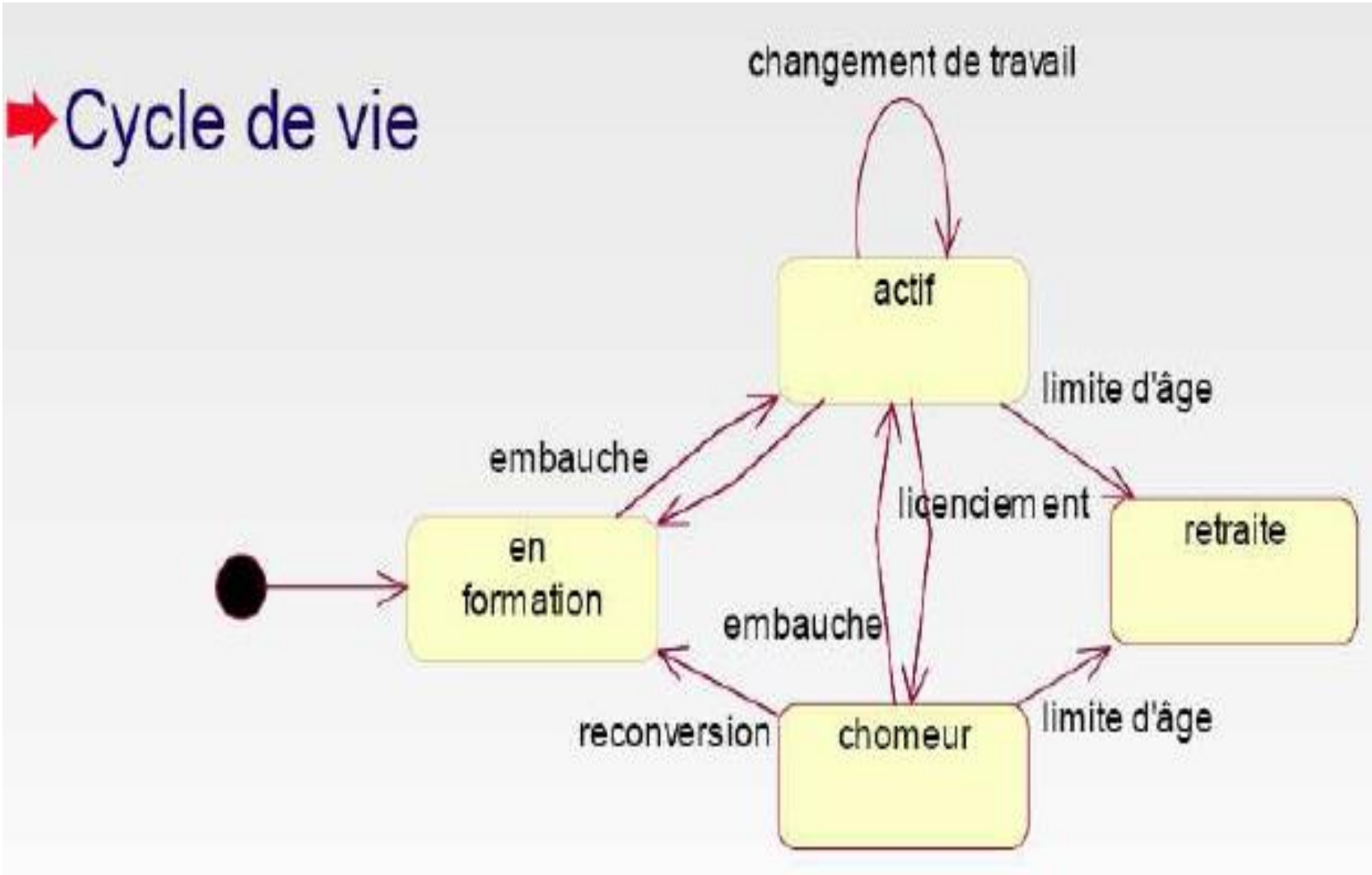




Transition d' achèvement

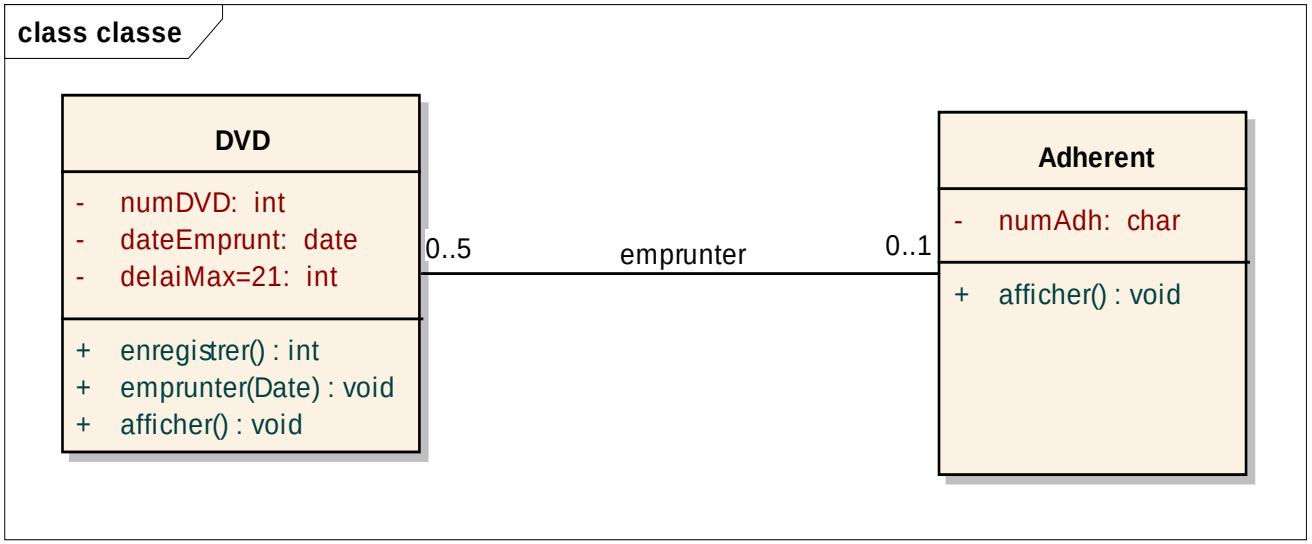
- Une transition dépourvue d'événement déclencheur explicite
- Se déclenche à la fin de l'activité contenue dans l'état source
- Peut contenir une condition de garde qui est évaluée au moment où l'activité contenue dans l'état s'achève.
- Utilisées par exemple pour connecter les états initiaux avec leur état successeurs.

Exemple 1


Cycle de vie
changement de travail
actif
limite d'âge
embauche
licenciement
retraite
en
formation
embauche
reconversion
chomeur
limite d'âge

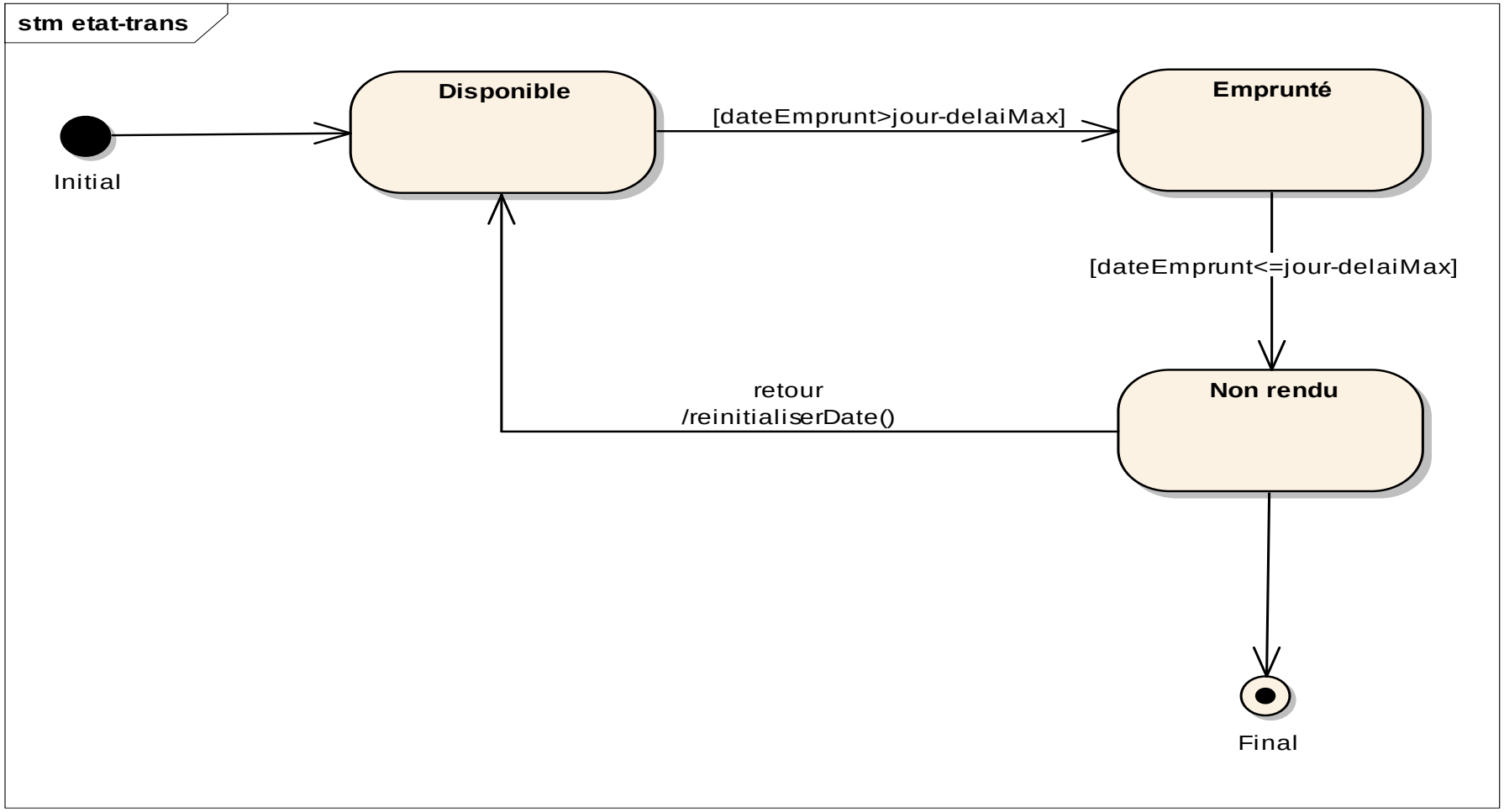
Exemple 2

- Modéliser par un diagramme d'états / transition les différents états de la classe DVD. Un DVD pouvant être ***disponible***, ***emprunter*** et ***non rendu***.

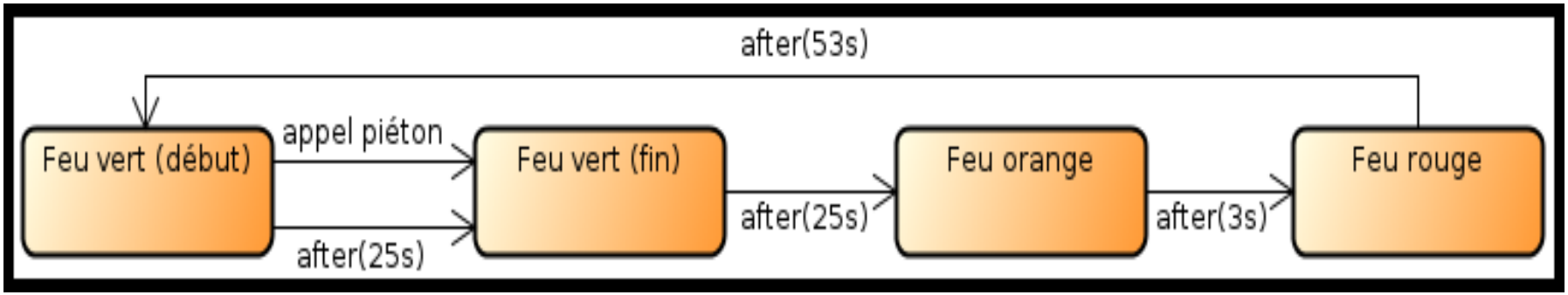


Exemple 2

Une représentation possible (selon les règles de gestion)



Exemple 3 : feu tricolore (avec un bouton appel piéton)



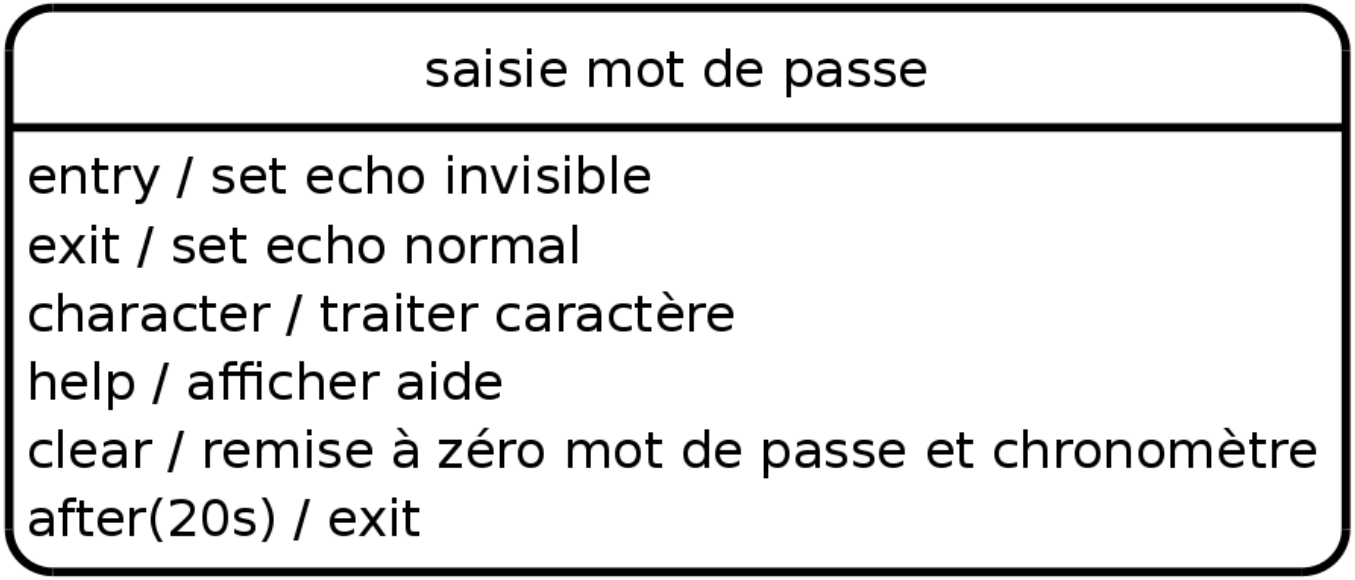
Transition interne

- Ses règles de déclenchement sont les mêmes que pour une transition externe
- Elle ne possède pas d'état cible
- L' état actif reste le même à la suite de son déclenchement.
- Ne sont pas représentées par des arcs mais sont spécifiées dans un compartiment de leur état associé.
- possèdent des noms d'événement prédéfinis correspondant à des déclencheurs particuliers : *entry*, *exit*, *do*, *include*, *on*

Transition interne

- **entry**/ : activité qui s'accomplit à l'entrée de l'état.
- **exit** /: activité qui s'accomplit à la sortie de l'état.
- **do** /: commence dès que l'activité entry est terminée (i.e pendant un état).
- **include** /: invoque un sous-diagramme d'états-transitions.
- **on env** /: au sein d'un état, lors de la réception d'un événement.

Transition interne exemple



Points de choix

- Représenter des alternatives pour le franchissement d'une transition :

le point de jonction (un petit cercle plein) :

- Peut avoir plusieurs segments de transition entrante
- Peut avoir plusieurs segments de transition sortante.
- Ne peut pas avoir des transitions sortantes dotées de déclencheurs d'événements

le point de décision (un losange) :

- Possède une entrée et au moins deux sorties.
- Les gardes situées après le point de décision sont évaluées au moment où il est atteint

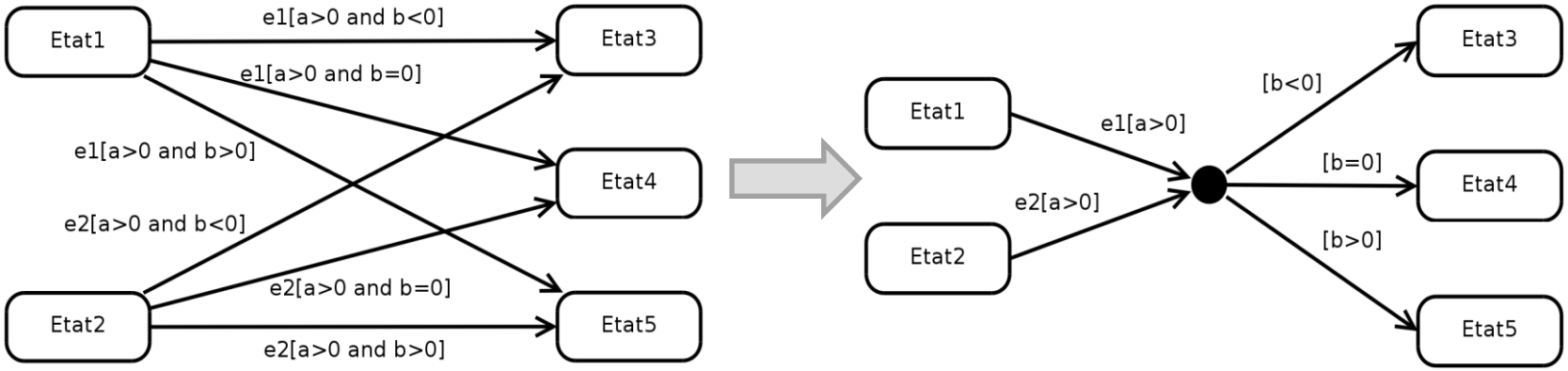


Diagramme d'activités

Présentation

- Permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.
- Ils mettent l'accent sur les traitements.
- Contrairement au diagramme d'états-transitions, ils ne sont pas spécifiquement rattachés à un classeur particulier.
- Sont particulièrement adaptés à la description des cas d'utilisation dans la phase de conception

Concepts utilises

- Action
- Activité
- Nœud d’ activités
- Transition
- Partition

Action

- Se rapproche de la notion d'instruction élémentaire d'un langage de programmation.

- C'est le plus petit traitement qui puisse être exprimé en UML.

- Une action peut être, par exemple :
  - *une **affectation de valeur** à des attributs ;*
  - *la **création d'un nouvel objet** ou lien ;*
  - *un **calcul arithmétique** simple ;*
  - ***l'émission d'un signal** ;*
  - *la **réception d'un signal** ;*

Activité

- Définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions.

- Le flot d'exécution est modélisé par des nœuds reliés par des arcs (transitions).

Nœud d'Activité

Un nœud d'activité est un élément permettant de représenter les étapes d'une activité.

Il existe trois familles de nœuds d'activités :

les nœuds d'exécutions ou exécutable : nœud d'activité qu'on peut exécuter.

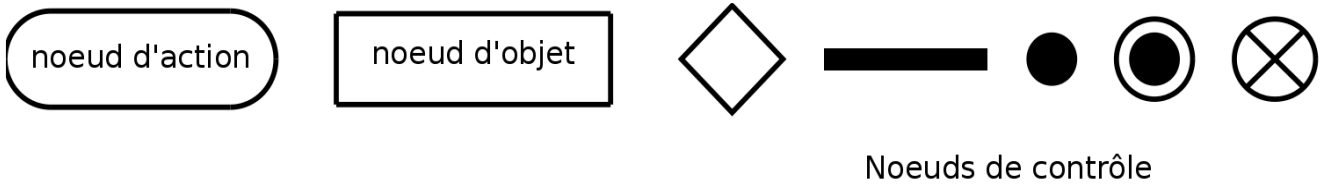
Les nœuds d'objets : représente l'existence d'un objet généré par une action dans une activité et utilisé par d'autres actions.

les nœuds de contrôle

Nœud d'Activité (suite)

les nœuds d'activité de la gauche vers la droite :

- ✓ Le nœud représentant une action,
- ✓ un nœud objet,
- ✓ un nœud de décision ou de fusion,
- ✓ un nœud de bifurcation ou d'union,
- ✓ un nœud initial,
- ✓ un nœud final
- ✓ un nœud final de flot.



Noeuds de contrôle

Transitions

✓Matérialise le passage d'un nœud d'activité vers un autre.

✓Sont représentées par des flèches en traits pleins qui connectent les activités (nœuds) entre elles.

✓Sont déclenchées dès que l'activité source est terminée

✓Provoquent automatiquement et immédiatement le début de la prochaine activité à déclencher (l'activité cible).

Partitions

✓ Sont aussi appelées couloirs d'activités ou lignes d'eau du fait de leur notation.

✓ Permettent d'organiser les nœuds d'activités dans un diagramme d'activités en opérant des regroupements.

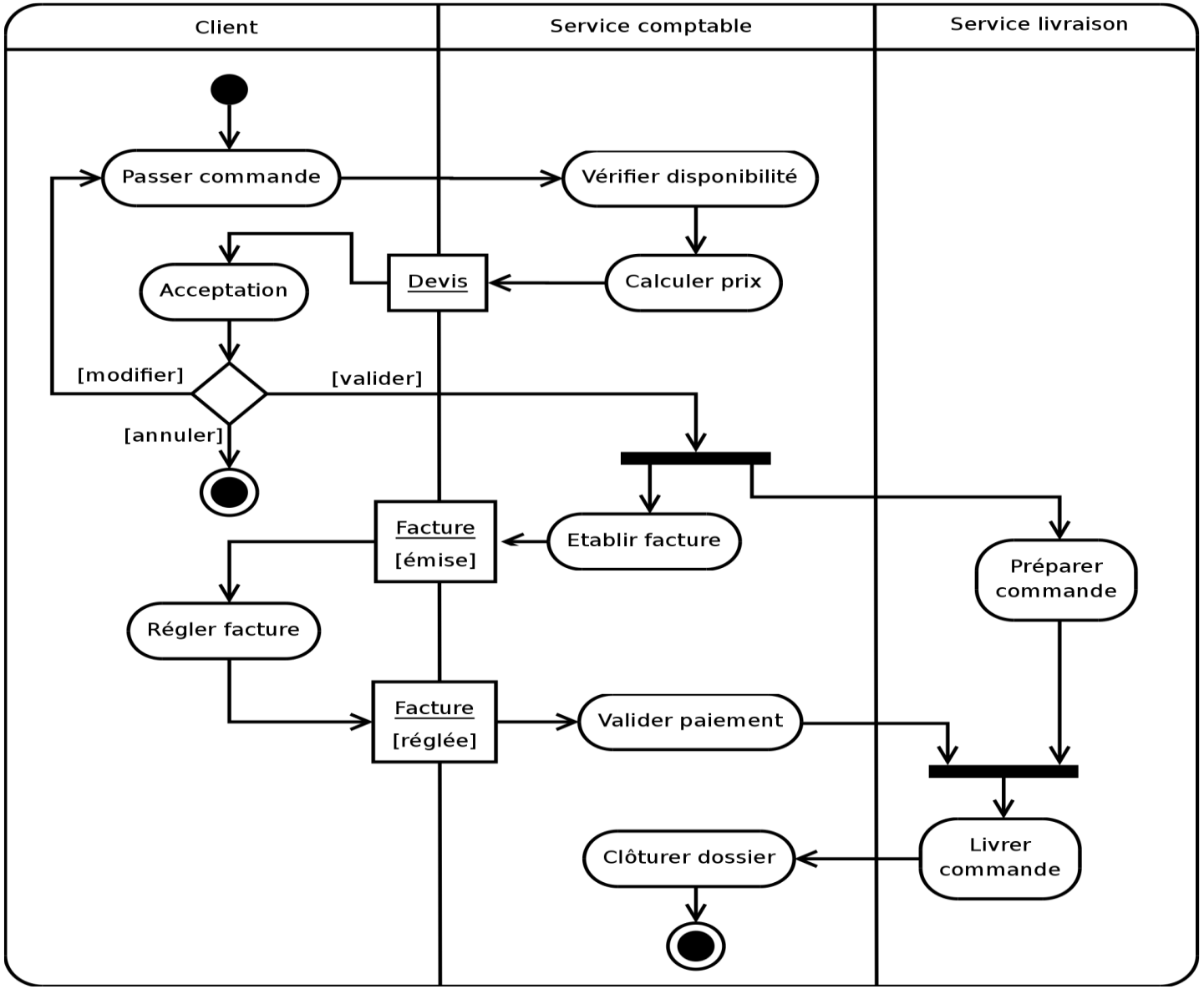
✓ On peut, par exemple, les utiliser pour spécifier la classe responsable de la mise en œuvre d'un ensemble de tâches.

✓ Dans ce cas, la classe en question est responsable de l'implémentation du comportement des nœuds inclus dans ladite partition.

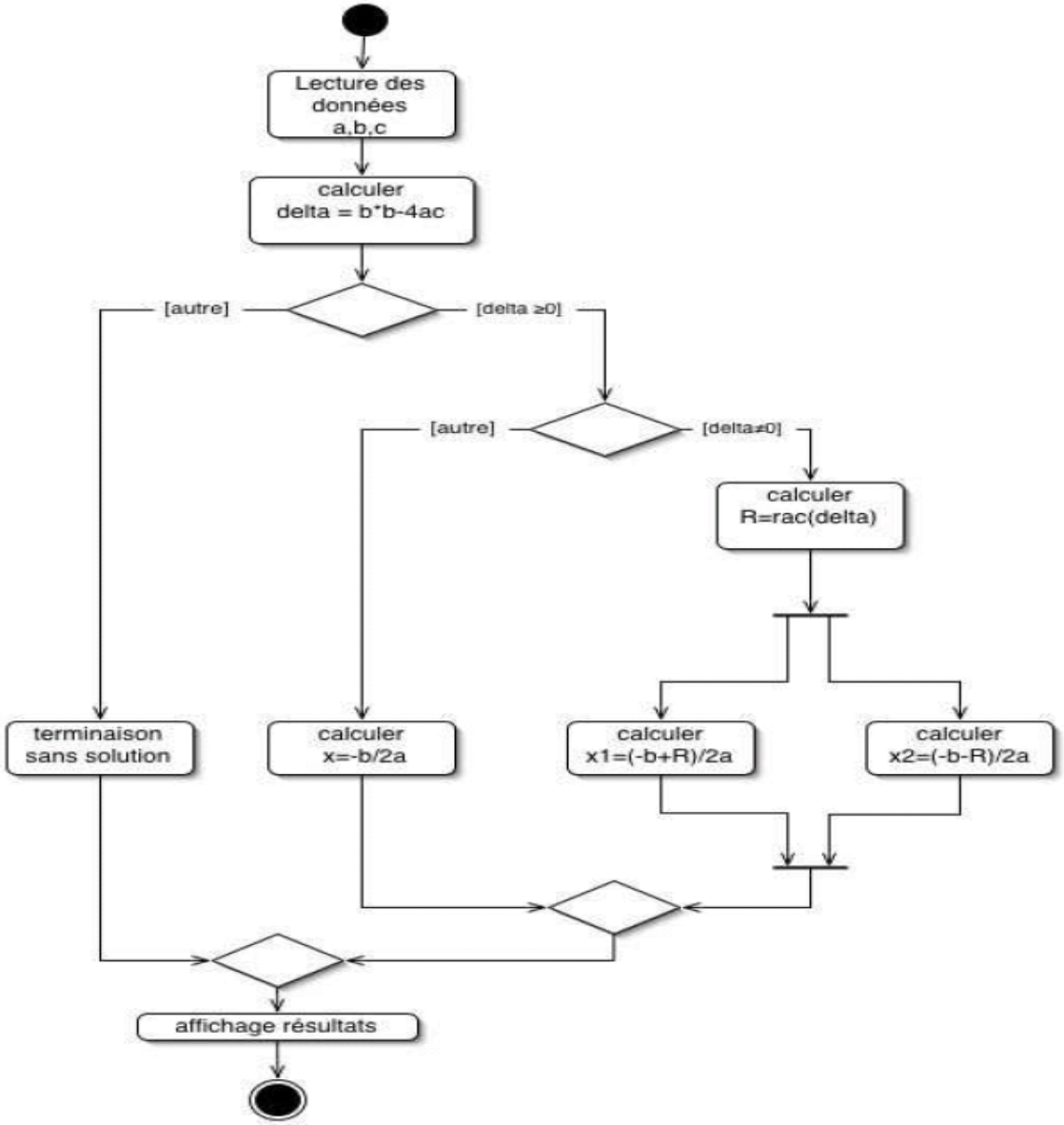
✓ Elles sont représentées par des lignes continues.

✓ Les transitions peuvent, traverser les frontières des partitions.

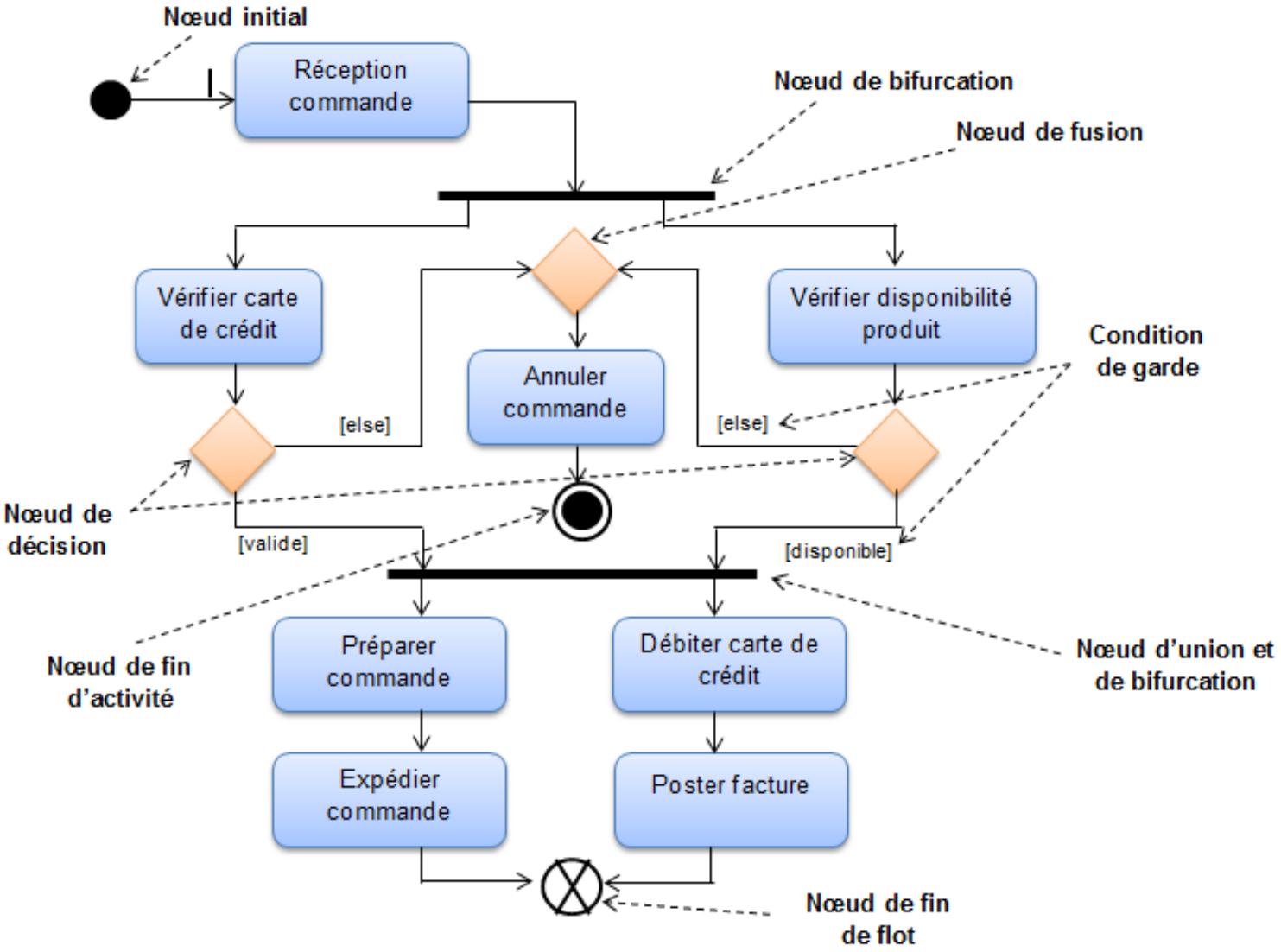
Représentation : exemple 1



Représentation : exemple 2



Représentation : exemple 3



https://fr.wikiversity.org/wiki/Mod%C3%A9lisation_UML/Le_diagramme_d'activités

Diagramme d’interaction

Présentation

- Diagramme d' état transition = se focalise sur un seul objet
- Diagramme d'interaction = fait intervenir une collection d'objets qui coopèrent
- Permettent de présenter les interactions entre les objet pour réaliser une tâche du système.
- Montrent comment des objets communiquent pour réaliser les fonctionnalités du système
- Permettent d'établir un lien entre les diagrammes de cas d'utilisation et les diagrammes de classes.
- Deux types de diagrammes d'interaction
  - ✓ Le **diagramme de communication** : met l'accent sur l'organisation structurelle des objets qui envoient et reçoivent des messages,
  - ✓ Le **diagramme de séquence** : met l'accent sur la chronologie de l'envoi des messages.

Diagramme de collaboration ou de communication

Présentation

- Permet de décrire la mise en œuvre d'une fonctionnalité par un jeu de participant

- **Exemple**: implémenter un cas d'utilisation = utiliser des classes + d'autres éléments, fonctionnant ensemble pour réaliser le comportement de ce cas d'utilisation.

- Cet ensemble d' éléments = structure statique + structure dynamique.

  est modélisé en UML par une collaboration.

- Utiliser pour illustrer un cas d'utilisation ou pour décrire une opération.

- Aide à valider les associations du diagramme de classe en les utilisant comme support de transmission des messages.

Concepts

- **Les participants :** sont représentés par des rectangles contenant une étiquette dont la syntaxe est : **[<nom_du_rôle>] : [<Nom_du_type>].** Au moins un des deux noms doit être spécifié.
- **Les connecteurs :** relations entre les représentants, se définissent par un trait plein.
- **Les messages :** sont généralement ordonnés selon un numéro de séquence croissant.
- Un message est, habituellement, spécifié sous la forme suivante :

['['<cond>']' [<séq>] :] [<var> :=] <msg>([<par>])

- ✓ <***cond***> : une condition sous forme d'expression booléenne entre crochets.
- ✓ <***séq***> : numéro de séquence du message. On numérote les messages par envoi et sous envoi désignés par des chiffres séparés par des points.
- ✓ <***var***> : valeur de retour du message, qui sera par exemple transmise en paramètre à un autre message.
- ✓ <***msg***> : le nom du message.
- ✓ <***par***> : les paramètres (optionnels) du message.

Exemple

Diagramme de communication illustrant la recherche puis l'ajout, dans un panier virtuel, d'un livre lors d'une commande sur Internet.

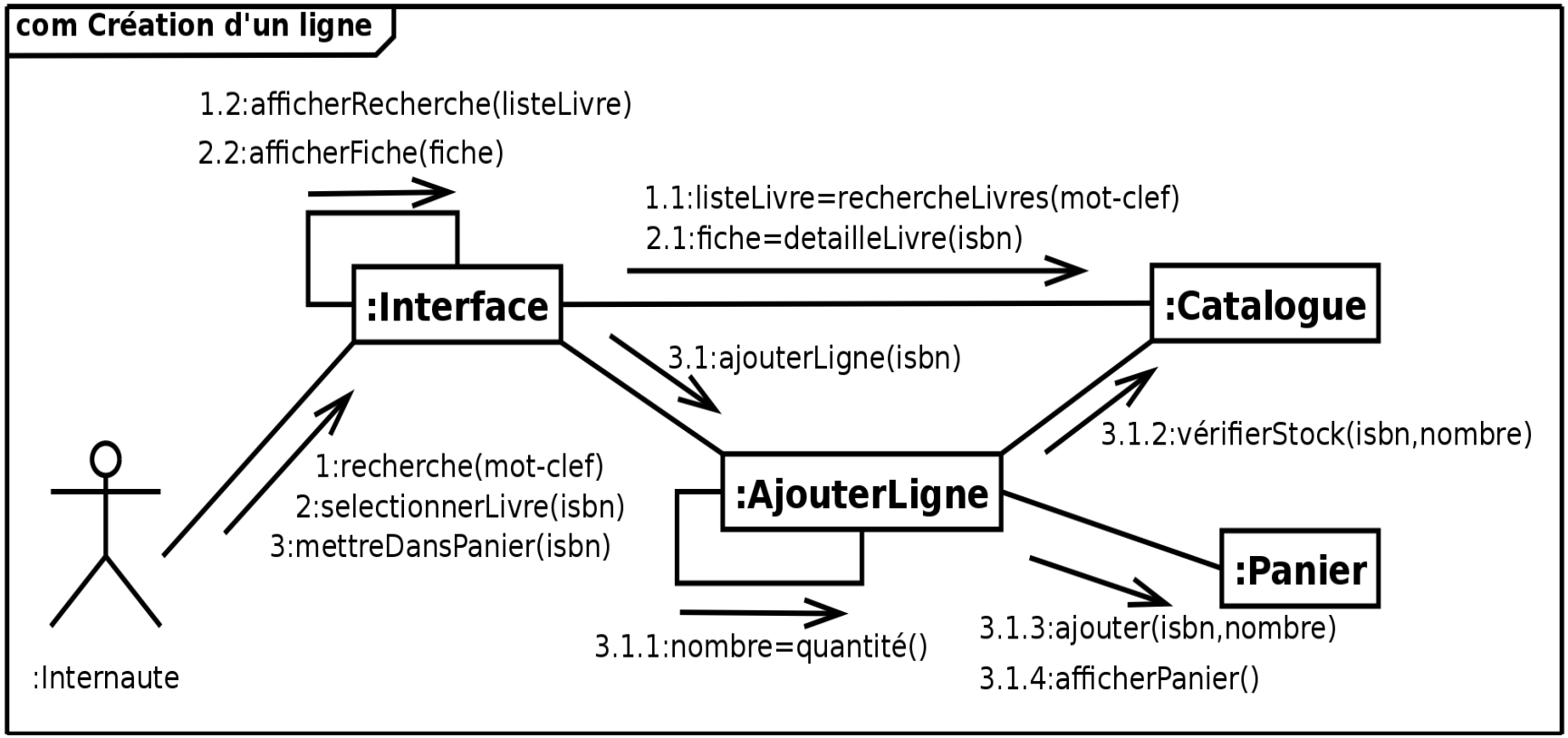


Diagramme de séquence

Présentation

- Principales informations contenu = messages échangés entre les lignes de vie,
- Messages présentés dans un ordre chronologique.
- Contrairement au diagramme de communication, le temps y est représenté explicitement par une dimension (La dimension verticale).
- Le temps s'écoule de haut en bas.

Concepts

- **Ligne de vie** :

  se représente par un rectangle, auquel est accrochée une ligne verticale pointillée, contenant une étiquette dont la syntaxe est : **[<nom_du_rôle>] : [<Nom_du_type>].**

- **Message** :

  Définit une communication particulière entre des lignes de vie.

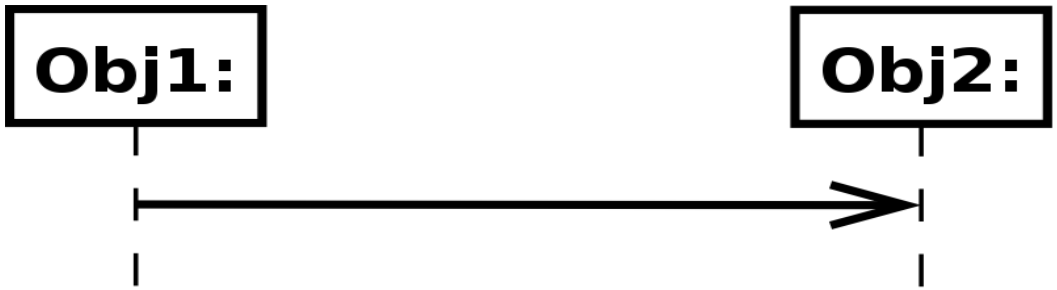
  Plusieurs types de messages parmi lesquels:
  – l'envoi d'un signal ;
  – l'invocation d'une opération;
  – la création ou la destruction d'une instance.

Types de messages

Message asynchrone:

N'attend pas de réponse et ne bloque pas l'émetteur qui ne sait pas si le message arrivera à destination ou s'il sera traité par le destinataire. Exemple : signal

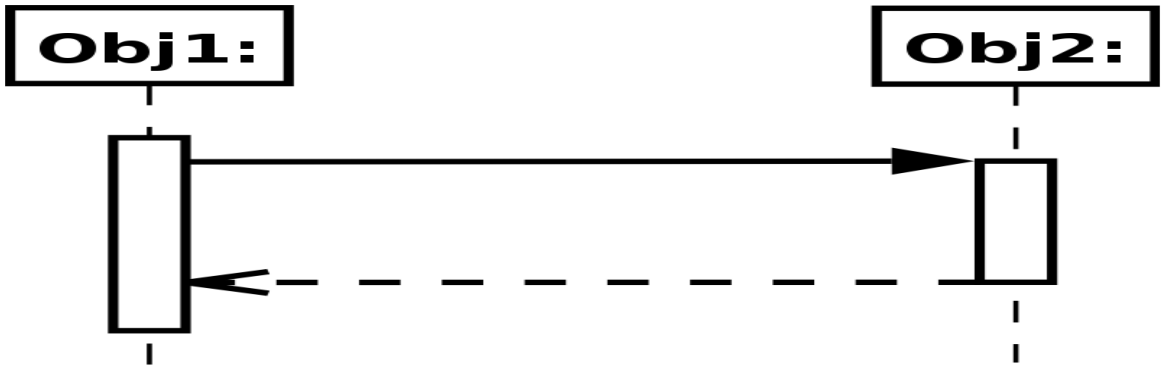
- Graphiquement, se représente par une flèche en traits pleins et à l'extrémité ouverte allant de la ligne de vie d'un objet vers celle d'un autre.



Types de messages

Message synchrone:

- L'émetteur reste bloqué le temps que dure le traitement du message.
- Graphiquement, se représente par une flèche en traits pleins et à extrémité pleine partant de la ligne de vie d'un objet vers celle d'un autre.
- Peut être suivi d'une réponse qui se représente par une flèche en pointillé.

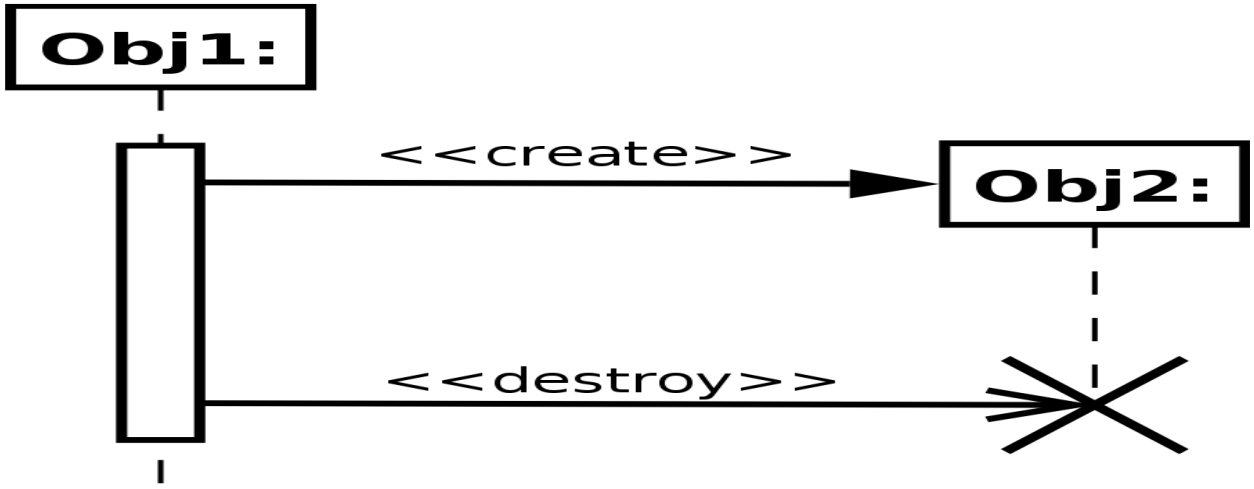


Types de messages

Message de création et destruction d'instance:

- La création d'un objet est matérialisée par une flèche qui pointe sur le sommet d'une ligne de vie.

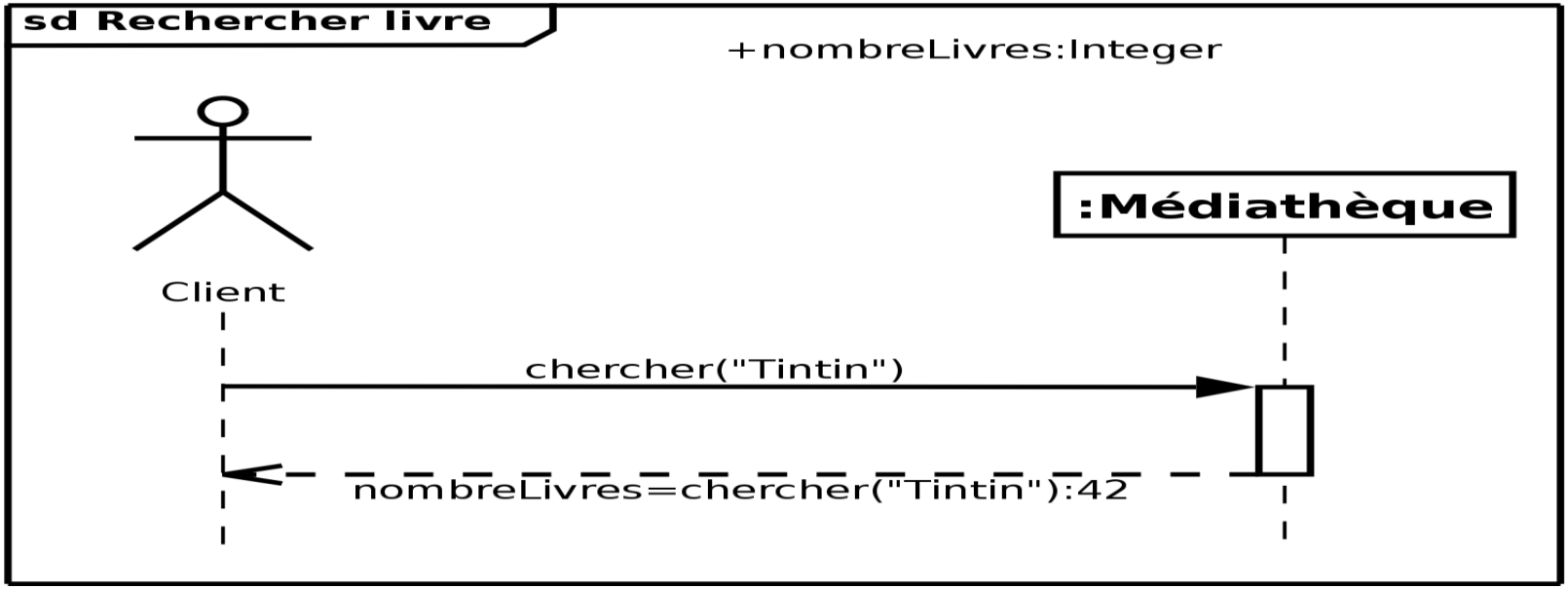
- La destruction est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet.



Syntaxe des messages et des réponses

- En générale, la réception d'un message est suivie de l'exécution d'une méthode d'une classe.
- Cette méthode peut recevoir des arguments
- La syntaxe des messages permet de transmettre ces arguments.
- La syntaxe de réponse à un message est la suivante :

[<attribut> =] message [: <valeur_de_retour>] , Où message représente le message d'envoi.



Fragments d'interaction combinés

- Un fragment combiné représente des articulations d'interactions, défini par un opérateur et des opérandes.
- L'opérateur conditionne la signification du fragment combiné.
- Un fragment est représenté dans un rectangle dont le coin supérieur gauche contient un pentagone.
- Dans le pentagone figure le type de la combinaison, appelé opérateur d'interaction.
- Les opérandes d'un opérateur d'interaction sont séparés par une ligne pointillée.
- Les conditions de choix des opérandes sont données par des expressions booléennes entre crochets.
- Il existe 12 operateurs d'interaction, parmi lesquels les opérateurs de choix et de boucle : ***alternative, option, loop, et break***.

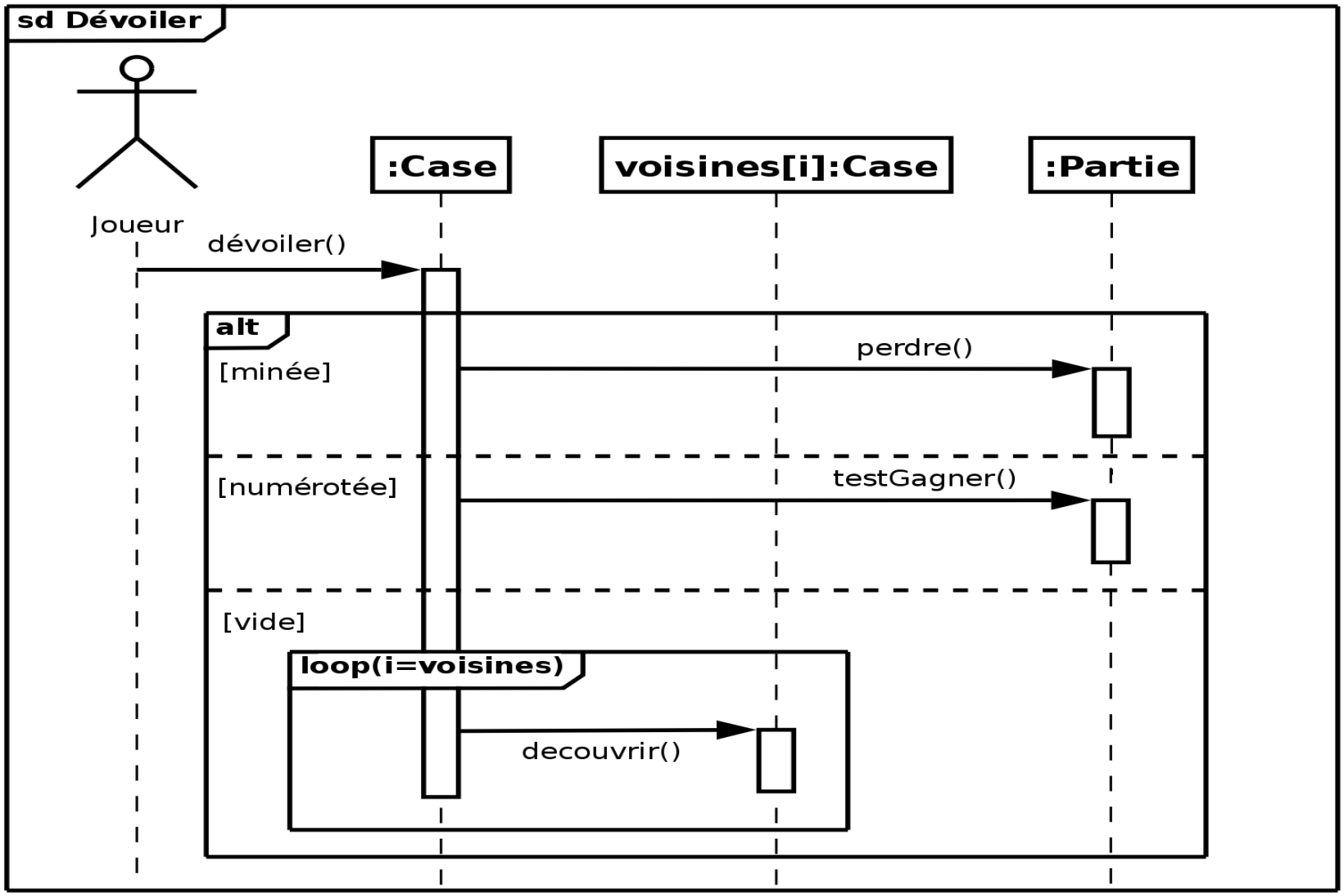
Fragments d'interaction combinés

Opérateur alt : opérateur conditionnel possédant plusieurs opérandes. C'est un peu l'équivalent d'une exécution à choix multiple (condition switch en C++).
La condition ***else*** est vraie si aucune autre condition n'est vraie. Exactement un opérande dont la condition est vraie est exécuté.

Opérateurs opt : comporte une opérande et une condition de garde associée.
Le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire.

Opérateur loop : possède un sous-fragment et spécifie un compte minimum et maximum (boucle) ainsi qu'une condition de garde.
loop['('<minInt> [',' <maxInt>] ')']
La condition de garde est placée entre crochets sur la ligne de vie. La boucle est répétée au moins ***minInt*** fois avant qu'une éventuelle condition de garde booléenne ne soit testée. Tant que la condition est vraie, la boucle continue, au plus ***maxInt*** fois.

Fragments d'interaction combinés



Fragments d'interaction combinés

L'opérateur ***ref***, permet de réutiliser une interaction

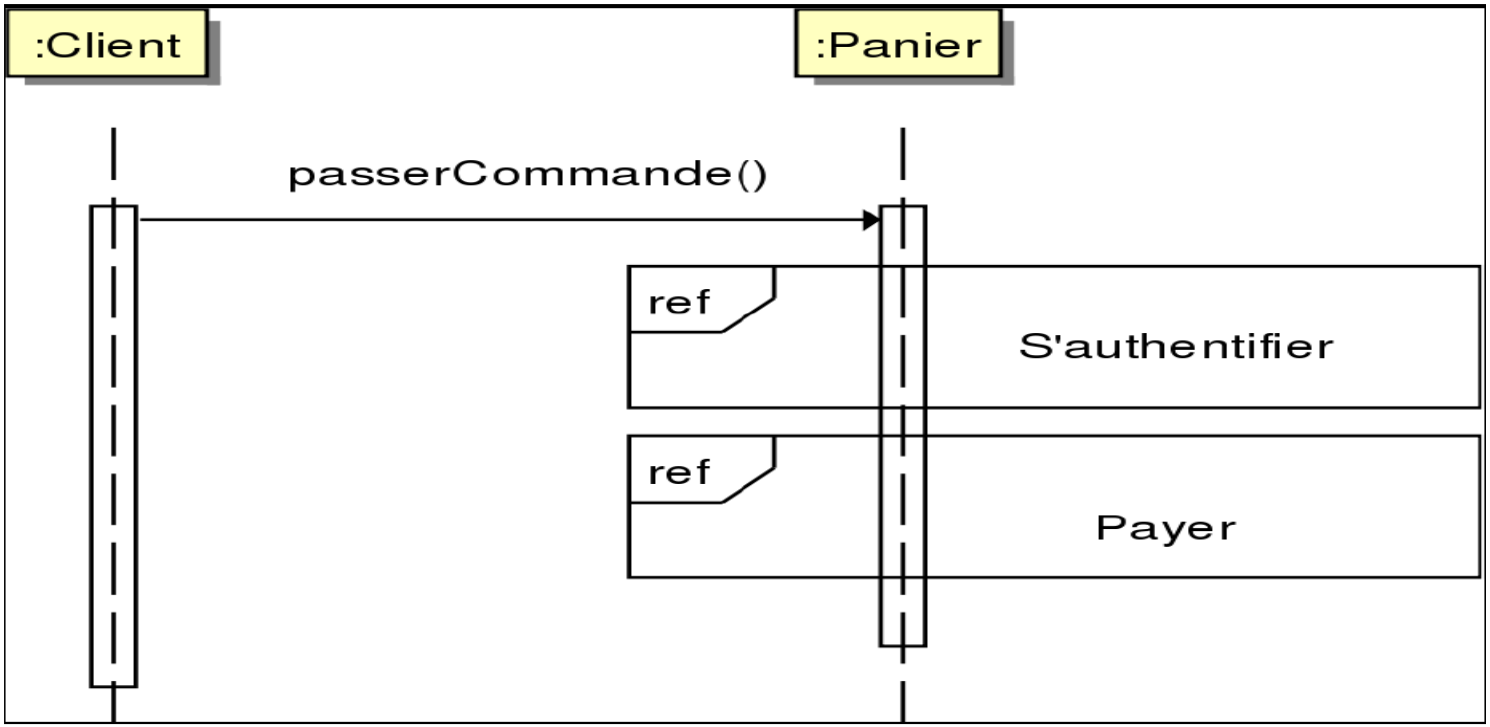


Diagramme de composants

Présentation

- La classe ne constitue pas une réponse adaptée à la problématique de la réutilisation : car faible granularité et connexions figées (liens structurels).
- Le diagramme de composants représente l'architecture logicielle du système.
- Ils permettent aussi de spécifier l'intégration de briques logicielles tierces (composants EJB, .Net, WSDL, etc.).
- Les composants peuvent contenir des classes.

Concepts

1. Composant

- Unité logicielle autonome représentée par un classeur stéréotypé «***Component***»;
- Comporte une ou plusieurs interfaces requises ou offertes;
- Son comportement interne est généralement réalisé par un ensemble de classes;
- Comportement totalement masqué : seules ses interfaces sont visibles.

2. Port

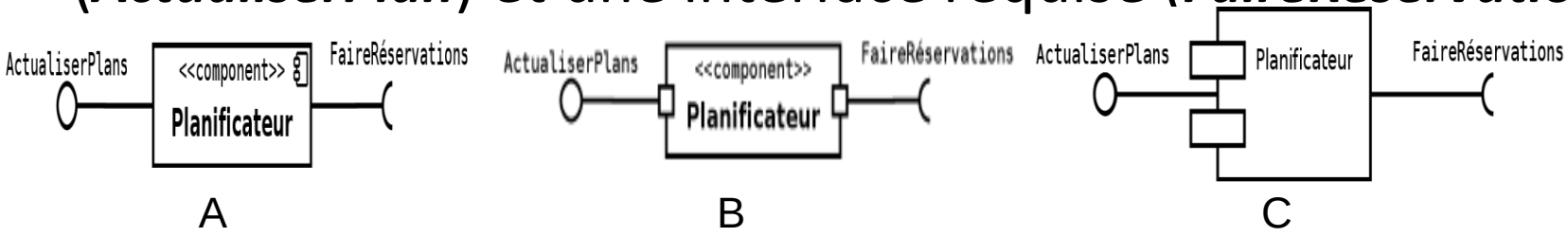
- Point de connexion entre un composant et son environnement.
- Graphiquement, est représenté par un petit carré à cheval sur la bordure du composant.
- On peut faire figurer le nom du port à proximité de sa représentation.
- Généralement, est associé à une interface requise ou offerte.

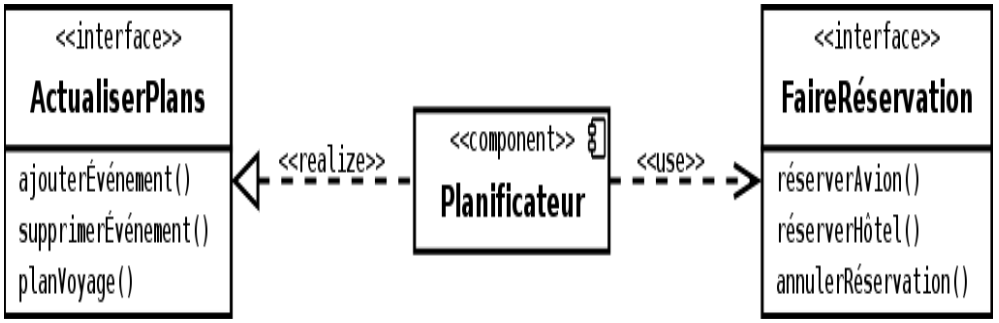
3. Interface

- Services fournis ou attendus par le composant

Représentation

- Les représentations A, B et C décrivent un composant nommé « **Planificateur** », qui a une interface fournie (***ActualiserPlan***) et une interface requise (***FaireReservation***).





Représentation : Exemple

Architecture logicielle d'une système de gestion des voyage.

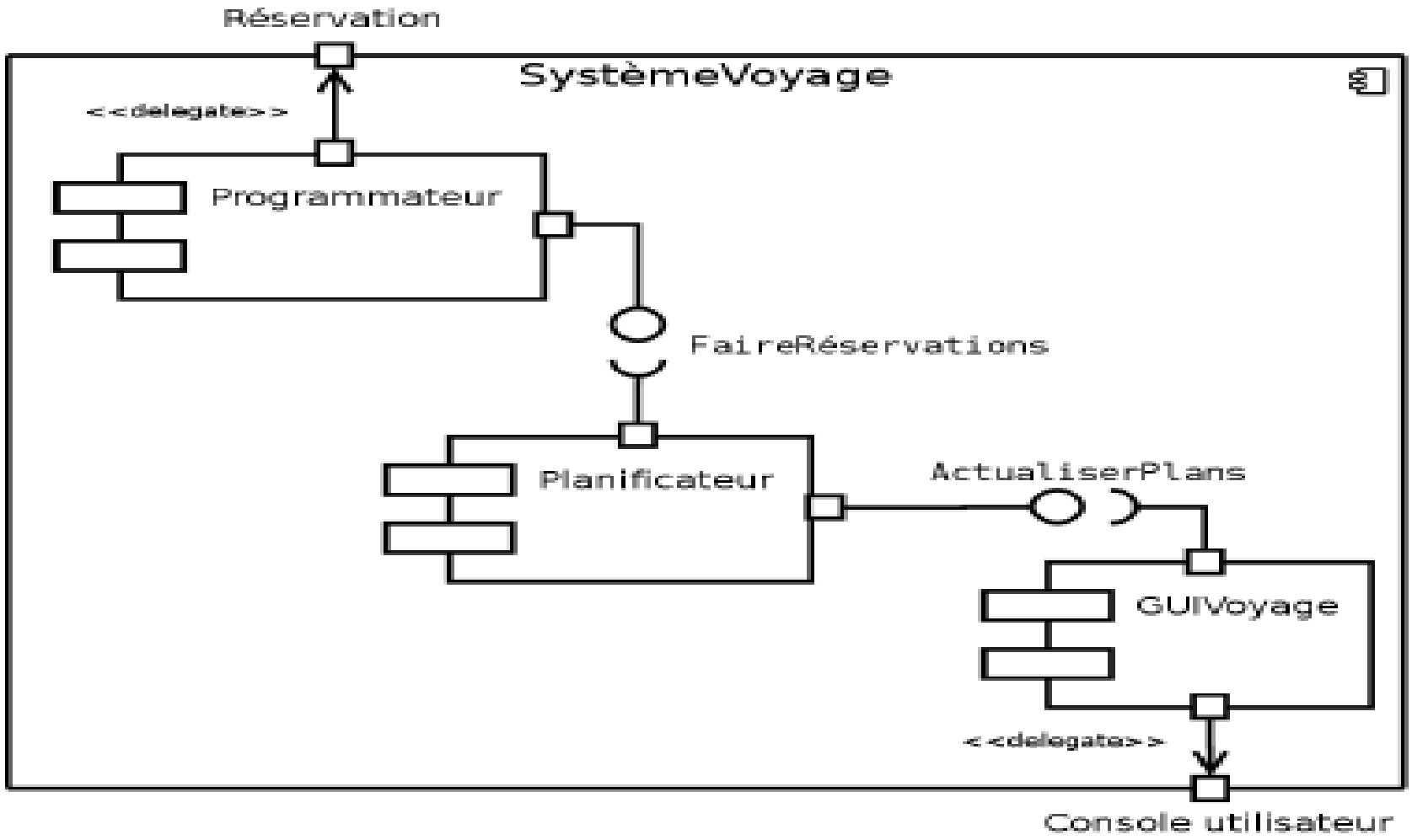


Diagramme de déploiement

Présentation

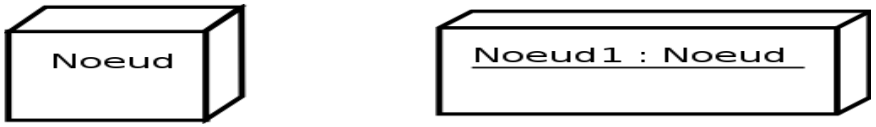
- Un diagramme de déploiement décrit la disposition physique des ressources matérielles qui composent le système;
- Montre la répartition des composants logiciels sur ces matériels;
- Montre les connexions entre ces composants.

- UML permet de représenter un environnement d'exécution ainsi que des ressources physiques grâce aux diagrammes de déploiement.

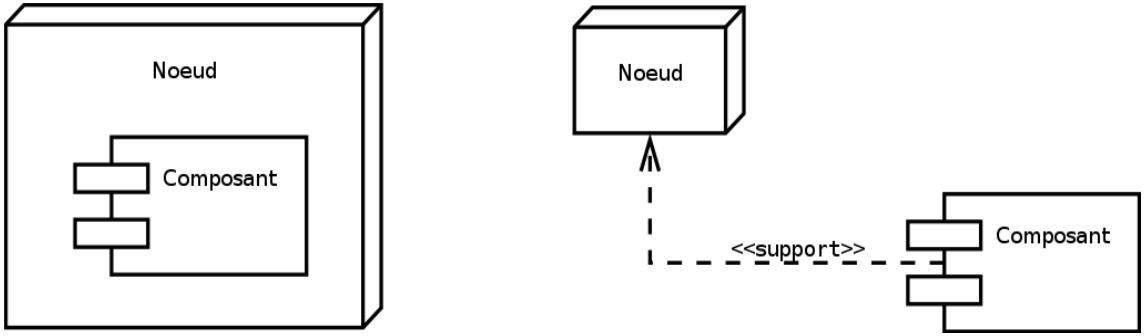
- Un nœud est une ressource sur laquelle des artefacts peuvent être déployés pour être exécutés.

Nœud

- Chaque ressource est matérialisée par un nœud représenté par un cube comportant un nom.
- Un nœud est un classeur et peut posséder des attributs (quantité de mémoire, vitesse du processeur, etc.).

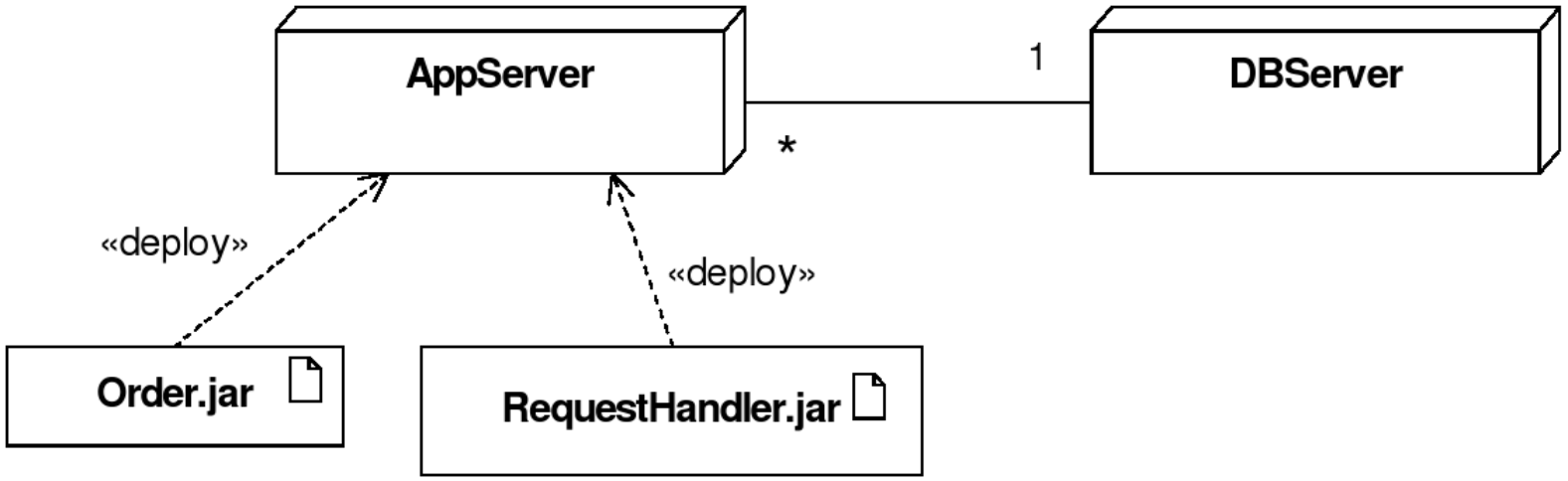


Une fois les nœuds définis, on doit leur affecter les composants. Pour montrer qu'un composant est affecté à un nœud, il faut soit placer le composant dans le nœud, soit les relier par une relation de dépendance stéréotypée «support» orientée du composant vers le nœud.



Artefact (artifact)

• Correspond à un élément concret existant dans le monde réel (document, exécutable, fichier, tables de bases de données, script, etc.).
• Il se représente comme un classeur par un rectangle contenant le mot-clé «***artifact***» suivi du nom de l'artefact.
• Une instance d'un artefact se déploie sur une instance de nœud. Graphiquement, on utilise une relation de dépendance stéréotypée «***deploy***» pointant vers le nœud en question. L'artefact peut aussi être inclus directement dans le cube représentant le nœud.



Exercices d'application

Dans un magasin, un commerçant dispose d'un système de gestion de stock d'articles, dont les fonctionnalités sont les suivantes :

1. Edition de la fiche d'un fournisseur.
2. Possibilité d'ajouter un nouvel article qui nécessite tout d'abord l'édition de la fiche fournisseur. Si le fournisseur n'existe pas, on peut alors le créer.
3. Edition de l'inventaire. Depuis cet écran, on a le choix d'imprimer l'inventaire, d'effacer un article ou d'éditer la fiche d'un article.

Exercices d'application

Pour chacun des énoncés suivants, donnez un diagramme des classes :

1. Tout écrivain a écrit au moins une œuvre
2. Les personnes peuvent être associées à des universités en tant qu'étudiants aussi bien qu'en tant que professeurs.
3. Un rectangle a deux sommets qui sont des points. On construit un rectangle à partir des coordonnées de deux points. Il est possible de calculer sa surface et son périmètre, ou encore de le translater.
4. Les cinémas sont composés de plusieurs salles. Les Films sont projetés dans des salles. Les projections correspondantes ont lieu à chacune à une heure déterminée.