Introduction à UML

CONSULTING BSTORM

Plan

1. Introduction à la Spécification
2. Aperçu d'UML
3. Diagramme de Use Case et Description Textuelle
4. Diagramme de Classes
5. Diagramme d'Activité
6. Diagramme de Séquence
7. Diagramme d'Etats
8. Autres Diagrammes

Introduction à la Spécification

CONSULTING BSTORM

Le cas de la bibliothèque universitaire

On vous demande de définir le système d'information qui sera capable de supporter le fonctionnement de la bibliothèque décrit dans le texte

Qu’est ce qu’une Spécification

Une spécification est la description des fonctions d'un logiciel en vue de sa réalisation

La spécification permet de définir ce que doit faire le système (le “QUOI”) et ne fait pas de prescription sur la manière dont le système réalisera ses fonctions (le “COMMENT”)

Objectifs d'une Spécification

L'objectif principal d'une spécification est de définir et communiquer la solution que l'on souhaite

2 niveaux de spécification:

- **Analyse:**
  - Analyser le domaine d'application du système
  - Comprendre les besoins
- **Conception:**
  - Concevoir le système en termes de classes et objets
  - Modéliser une solution

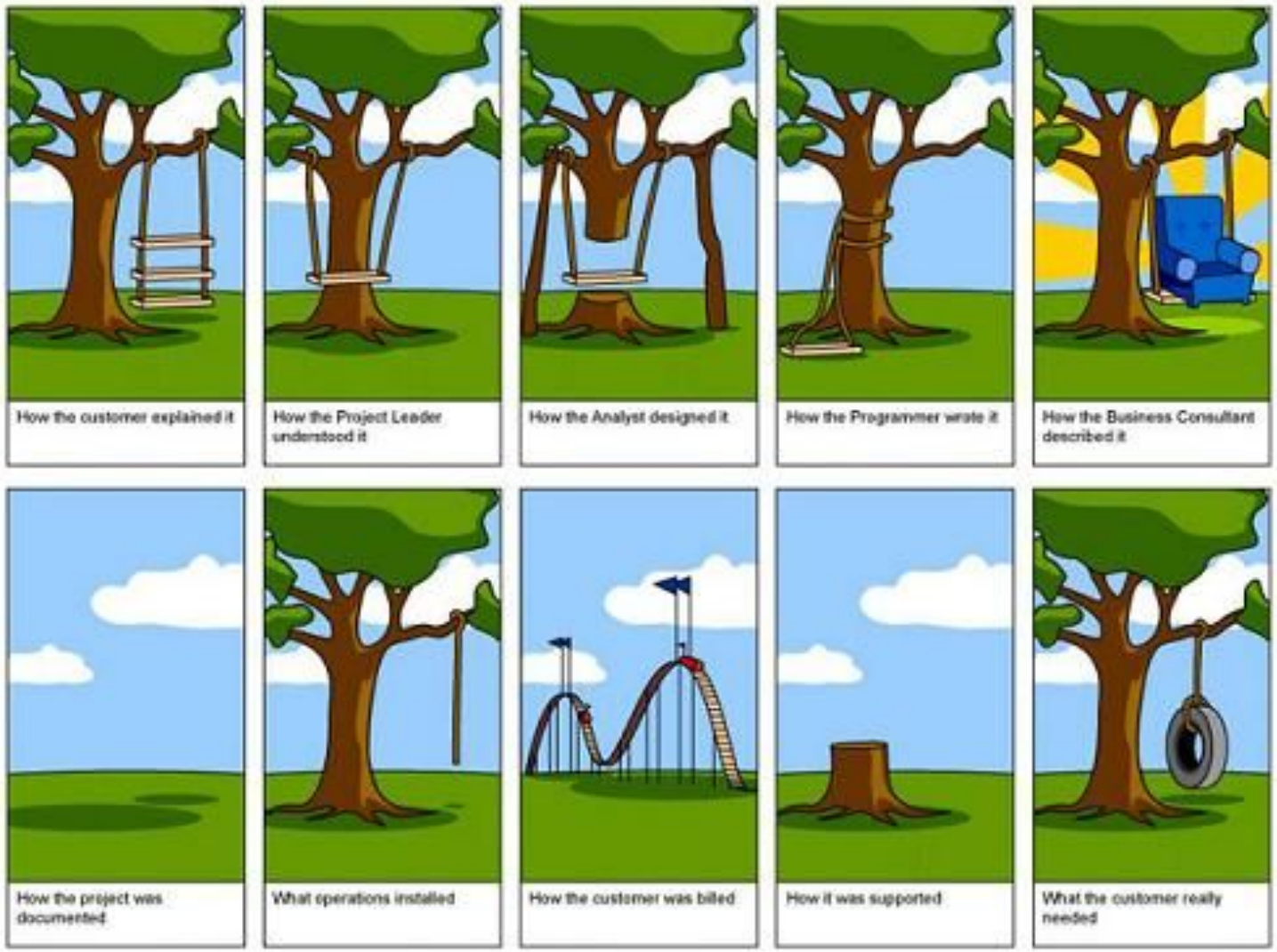
Exemple de Spécification

Votre manager vous envoie par mail la spécification qu'il a élaborée suite à son entretien avec le client.

Il vous est demandé de:

> "Dessiner une pizza qui a 8 parts avec 3 traits"

Problème de la communication



3 Types de Langages de Spécification

- Langage naturel
- Langage semi-formel
- Langage formel

Langage Naturel

- Définition:
  - Langue « normale » parlée par un être humain
- Avantages:
  - Pas (peu) d'apprentissage
- Inconvénients:
  - Ambigu
  - Manque de visibilité
  - ...
- Exemples:
  - Français utilisé dans les descriptions textuelles des Cas d'Utilisation

Exemple Spécification Langage Naturel

Use Case 2 **Get Paid for Car Accident**

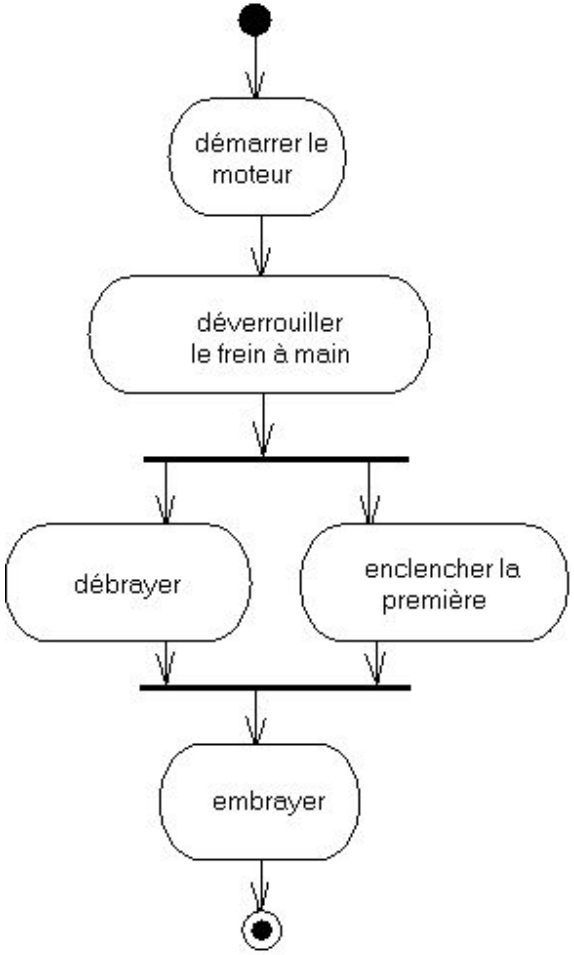
Primary Actor: Claimant
Scope: Insurance company ("MyInsCo")
Level: Summary
Stakeholders and Interests:
Claimant—to get paid the most possible.
MyInsCo—to pay the smallest appropriate amount.
Department of Insurance—to see that all guidelines are followed.
Precondition: None.
Minimal Guarantees: MyInsCo logs the claim and all activities.
Success Guarantees: Claimant and MyInsCo agree on amount to be paid; claimant gets paid that.
Trigger: Claimant submits a claim.
Main Success Scenario:
1. Claimant submits claim with substantiating data.
2. Insurance company verifies claimant owns a valid policy.
3. Insurance company assigns agent to examine case.
4. Insurance company verifies all details are within policy guidelines.
5. Insurance company pays claimant and closes file.

Extensions:
1a. Submitted data is incomplete:
1a1. Insurance company requests missing information.
1a2. Claimant supplies missing information.
2a. Claimant does not own a valid policy:
2a1. Insurance company denies claim, notifies claimant, records all this, terminates proceedings.
3a. No agents are available at this time.
3a1. (What does the insurance company do here?)
4a. Accident violates basic policy guidelines:
4a1. Insurance company denies claim, notifies claimant, records all this, terminates proceedings.
4b. Accident violates some minor policy guidelines:
4b1. Insurance company begins negotiation with claimant as to amount of payment to be made.

Langage Semi-Formel

- Définition:
  - Langage avec une syntaxe précise mais sans sémantique précise et non ambiguë
- Avantages:
  - Souvent facile à apprendre
  - Facilite une interpretation commune
  - Facilite la communication
- Inconvénients:
  - Ne lève pas toutes les ambiguïtés
- Exemples:
  - BPMN, UML

Exemple Spécification Semi-Formelle



Langage Formel

- Définition:
  - Langage avec une syntaxe et une sémantique précise
- Avantages:
  - Non ambiguité
- Inconvénients:
  - Difficile à apprendre
  - Expressivité souvent réduite
- Exemples:
  - Logique des prédicat, logique temporelle, Z
  - Code: PHP, C++, C#, Java...

Exemple Spécification Formelle

Professor

↾ (New, Affiliate)

Id: Word
Name: ℙ Word
Expertise: ℙ Word
Faculty: ℙ Word

New

Δ (Id, Name, Expertise)
i?: Word
n?: ℙ Word
e?: ℙ Word

Id′ = i? ∧ Name′ = n? ∧ Expertise′ = e?

Affiliate

Δ (Faculty)
f?: ℙ Word

Faculty′ = f?

Academician

↾ (New)

Id: Word
Name: ℙ Word
Expertise: ℙ Word
Faculty: ℙ Word

New

Δ (Id, Name, Expertise, Faculty)
i?: Word
n?: ℙ Word
e?: ℙ Word
f?: ℙ Word

Id′ = i?
Name′ = n?
Expertise′ = e?
Faculty′ = f?

TeachingStaff

↾ (Add, Join)

Id: Word
Name: ℙ Word
Institution: ℙ Word

Add

Δ (Id, Name)
i?: Word
n?: ℙ Word

Id′ = i?
Name′ = n?

Join

Δ (Institution)
f?: ℙ Word

Institution′ = f?

Objectifs de la formation

Introduction à la spécification

- Comprendre l'importance d'une spécification
- Savoir rédiger une spécification

Introduction à la conception OO

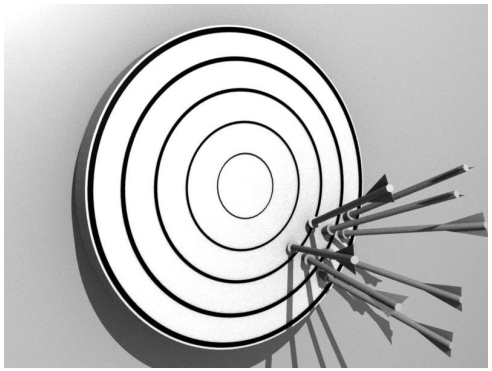
- Comprendre les mécanismes OO
- Pouvoir structurer un programme en termes de classes et d'objets

Approche de la formation

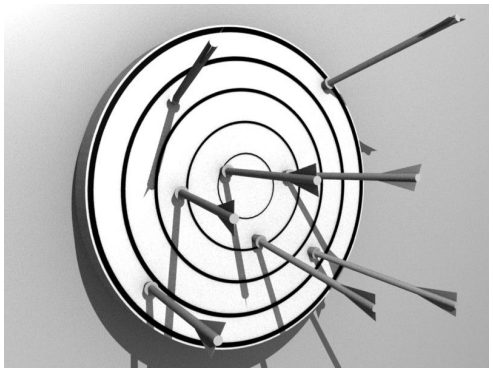
Un bon diagramme UML n'est pas forcément le plus détaillé

Il faut adapter le niveau de détails en fonction de l'objectif de la modélisation (analyse/conception)

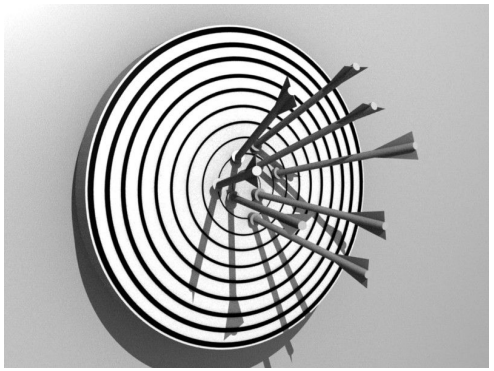
Justesse et Précision



Précis mais biaisé

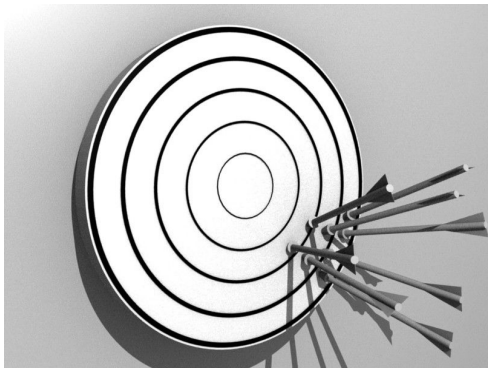


Imprécis mais juste

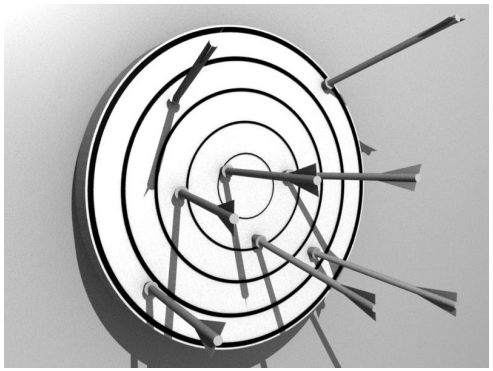


Précis et juste

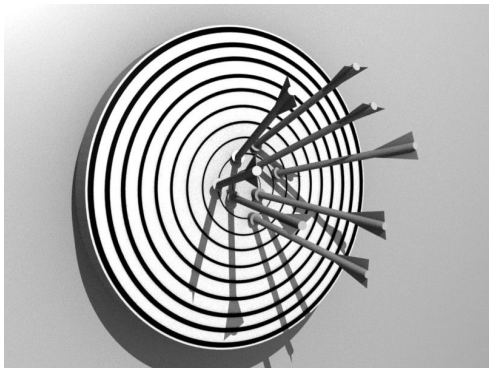
Justesse et Précision



Précis mais biaisé



Imprécis mais juste



Précis et juste

Quid du plus prioritaire?

Justesse et Précision

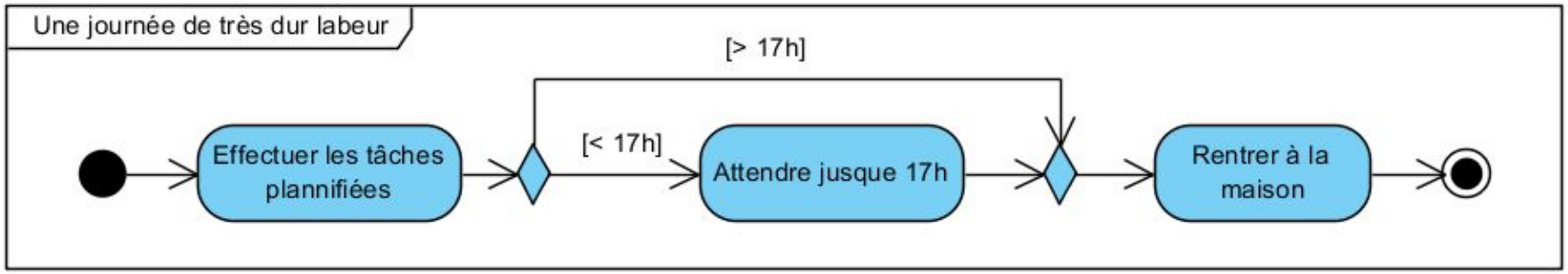
Quelle valeur de Pi vaut-il mieux utiliser ?

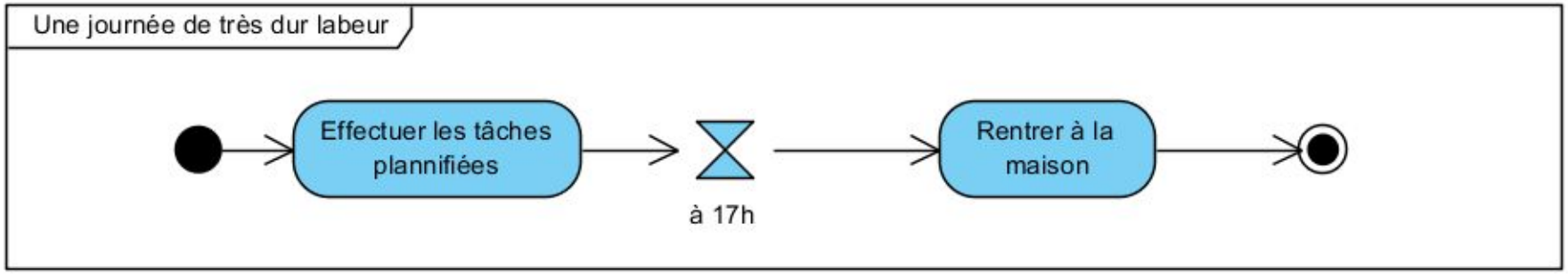
4.14159265358979323846264
(précise mais pas juste)

ou

3.1
(moins précise mais plus juste)

Justesse et Précision





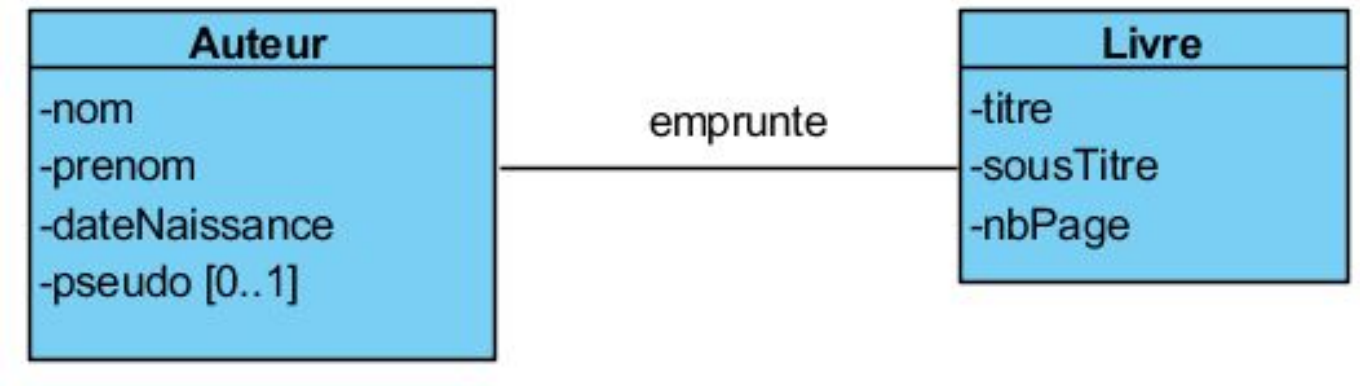
Correction et Cohérence

Quand un diagramme est-il bon?

- **Correct:**
  - Respecte les règles du langage (de modélisation)
- **Cohérent:**
  - Modéliser adéquatement le domaine d'application

Exemple diagramme

Correct ? Cohérent ?



Exemple diagramme

Correct ? Cohérent ?

Exemple diagramme

Correct ? Cohérent ?

Complétude

Est-il possible de décrire **complètement** ce bâtiment au travers d'un plan unique ?

Complétude

Est-il possible de décrire **complètement** ce bâtiment au travers d'un plan unique ?

Vue de l'architecte

Vue de l'électricien

Vue du cadastre

Vue de l'urbaniste

Vue du client

Aperçu d'UML

CONSULTING BSTORM

Les différentes perspectives

- Perspective informationnelle
  - Contenu: les données
  - Diagramme de classes UML (autre: Entité-Association)
- Perspective fonction
  - Contenu: les services, tâches, opérations,…
  - Diagramme de Use Case UML (autre: Features Diagrams)
- Perspective dynamique
  - Contenu: le comportement des objets, acteurs, document,…
  - Diagramme de séquence, de collaboration, d'état et d'activité UML (autre: les réseaux de Petri)
- Perspective de l'Orienté Objet
  - Contenu: données + fonctions (dynamisme!)
  - Diagramme de Classe UML

Acronyme UML

- **U**nified
  - Ensemble de diagramme (complétude)
- **M**odeling
  - Modèle, vue abstraite et cohérente d'une problématique
- **L**anguage
  - Outils de communication semi-formel
  - Définit des règles de construction (correction)

Les différents diagrammes

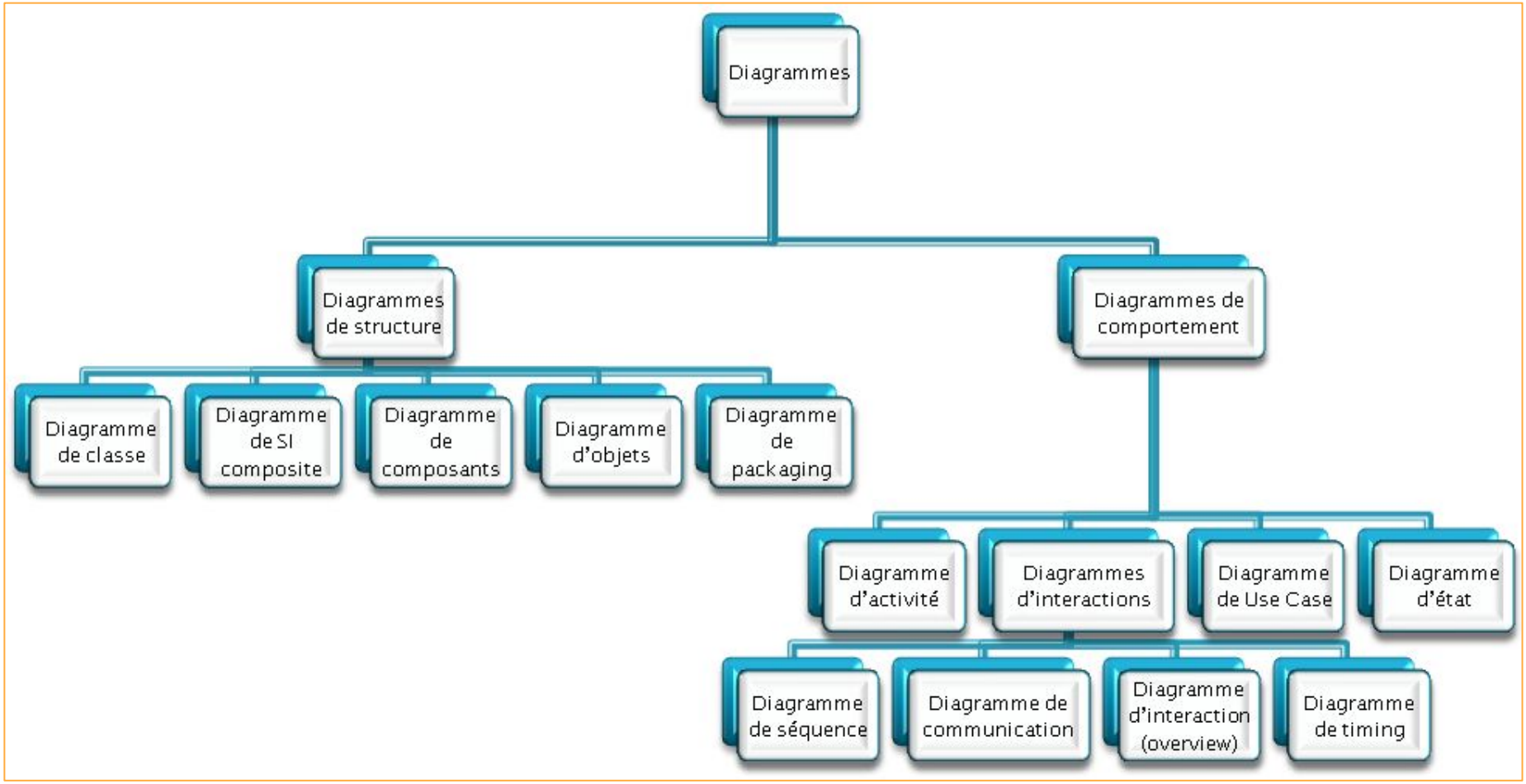


Diagramme de Use Cases

CONSULTING BSTORM

Motivation

- Quelles sont les différents cas où le système va être utilisé ?
  - Emprunter un livre, s'inscrire…
- Quels sont les acteurs qui interagissent avec le système ?
  - Les membres de l'UNamur…
- Quels sont les liens entre les acteurs et les cas d'utilisation ?
  - Les membres peuvent emprunter un livre

Définition

Recueillir et décrire les besoins des acteurs du système

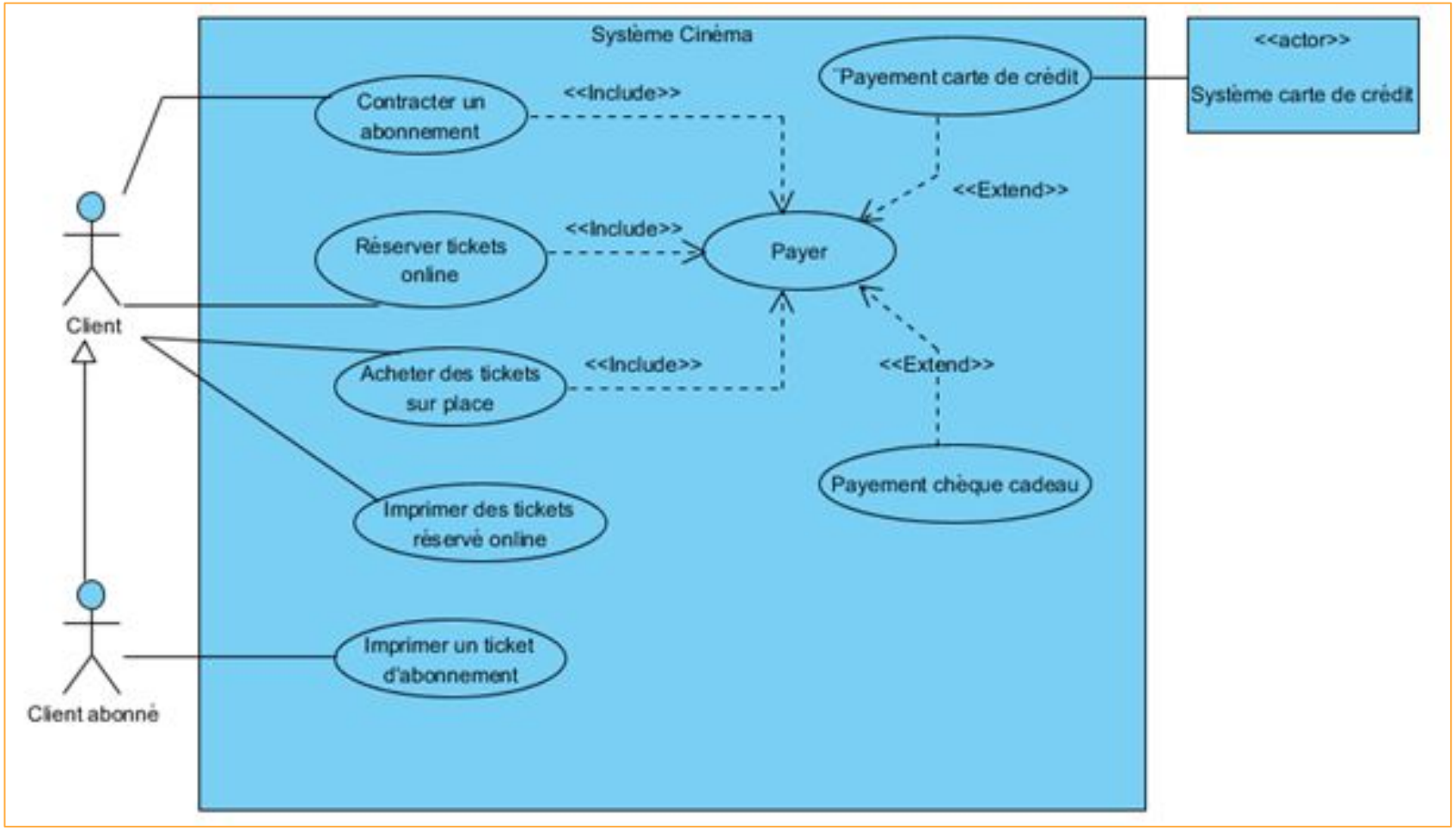
Objectifs et Contexte d'Utilisation

Analyse (Business)

- Chaque cas d'utilisation permet de décrire l'interaction entre le système et différents acteurs en vue de satisfaire un objectif d'un des acteurs

- Les Use Cases décrivent comment le système se comporte et réagit avec l'extérieur sans révéler la manière dont le système est construit

Exemple


Système Cinéma
Contracter un abonnement
<<Include>>
Payement carte de crédit
<<actor>>
Système carte de crédit
<<Extend>>
Réserver tickets online
<<Include>>
Payer
Client
Acheter des tickets sur place
<<Include>>
<<Extend>>
Payement chèque cadeau
Imprimer des tickets réservé online
Imprimer un ticket d'abonnement
Client abonné

Overview

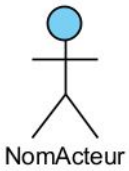
Le diagramme de Use Case est le diagramme **le plus simple** mais souvent **le plus mal utilisé**

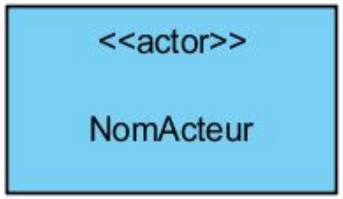
- 3 concepts
  - Acteurs
  - Cas d'utilisation
  - 4 types de relations
- Peu de concepts mais peu (ou pas) de consensus sur leur sémantique
- Doit être lié à des descriptions textuelles:
  - Pas de standard sur la manière de rédaction

Acteur

Un acteur est un rôle présentant toujours le même comportement vis-à-vis du système

- Un acteur est un agent externe au système
- Un acteur peut être un humain ou autre système
- Représentation:





Acteur

- Il existe différent type d'acteur:
  - **Le système**: le système en lui-même est un acteur à part entière (représenté par *"la frontière du système"*)
  - **Acteur primaire**: acteur dont l'objectif est satisfait par l'exécution du Use Case (souvent mais pas toujours l'initiateur du Use Case)
  - **Acteur secondaire**: acteur qui fournit un service de support au système durant l'exécution d'un Use Case
  - **Acteurs internes au système (white-box)**:
    N'apparaissent que dans la description textuelle de use case au niveau business

Acteur

Bonnes Pratiques:

- La dénomination d'un acteur doit contenir son rôle et/ou responsabilités
- La bonne représentation pour le bon rôle
  - Stickboy pour les acteurs humains
  - Rectangle (Classifier) pour les systèmes non-humains

Acteur

Etude de cas - Bibliothèque:

Identifier les différents acteurs du système

Acteur

Etude de cas - Bibliothèque:

Quid de la bibliothécaire?

Use Case

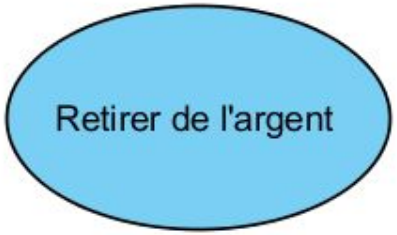
Un Cas d'Utilisation est la description de la séquence possible d'interactions entre le système et ses acteurs extérieurs, en relation à un objectif particulier [A. Cockburn]

- Chaque séquence possible d'interaction est appelé un scénario (décrit textuellement dans un document)
- Le Cas d'Utilisation regroupe tous les scénarios en relation avec l'objectif du Use Case, que ce dernier soit satisfait ou non
- Représentation:

Use Case

Bonnes Pratiques:

- La dénomination doit être un verbe infinitif suivi d'un groupe nominal



Use Case

Etude de cas - Bibliothèque:

Identifier les principaux cas d'utilisation du système

Description textuelle

Nom du use case

Description globale (4-5 lignes)

Acteurs principaux - secondaires

Pré-conditions globales

Post-conditions globales

Happy Scénario

- Pré et Post spécifiques
- Scénario

Cas alternatif 1

Pré et Post spécifiques

- Scénario

Cas alternatif 2

- ...

1. X fait...
2. Y demande à X...
3. Z valide ...
4. Y fait ceci
5. X fait cela
6. ...

Use Case

Etude de cas - Bibliothèque:

Réaliser la description d’un cas d’utilisation

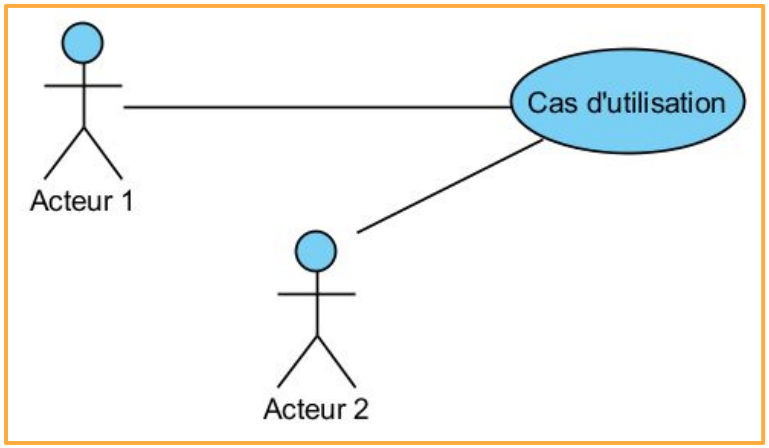
Relations

4 Types de Relations:

- Association
- Include
- Extend
- Spécialisation/Généralisation

Association

- Permet de décrire les échanges **entre un acteur et un cas d'utilisation**
- Multiplicité: par défaut 0..1
- Représentation:



Association

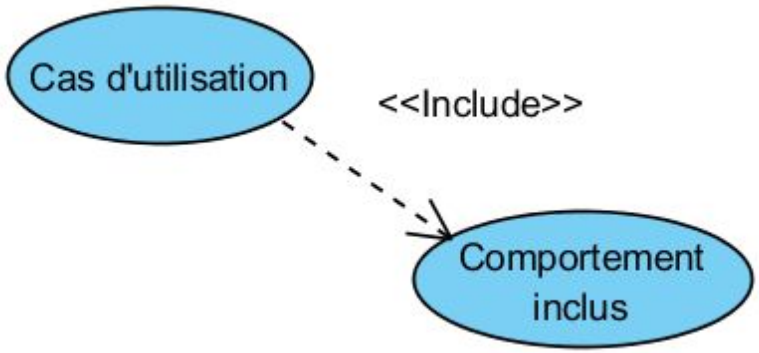
Etude de cas - Bibliothèque:

Lier les différents acteurs avec les différents cas d'utilisation du système

Include

- Permet de décrire qu'un cas d'utilisation contient **inconditionnellement** un comportement décrit dans le "cas d'utilisation" inclus

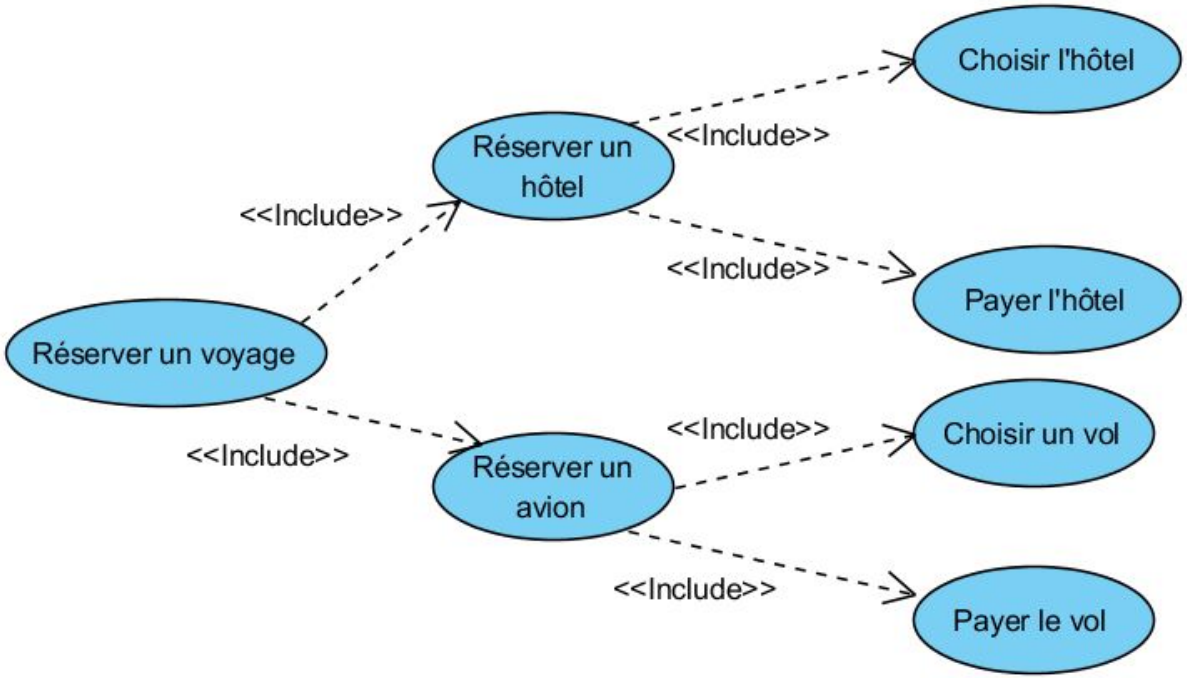
- Représentation:



Include

Bonnes pratiques

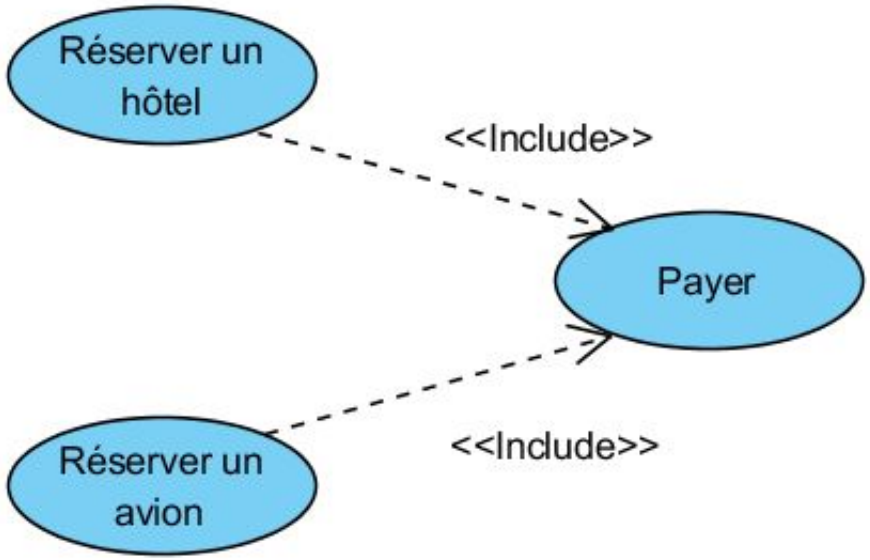
- Ne **jamais** utiliser la relation d'include pour faire de la *décomposition fonctionnelle*



Include

Bonnes pratiques

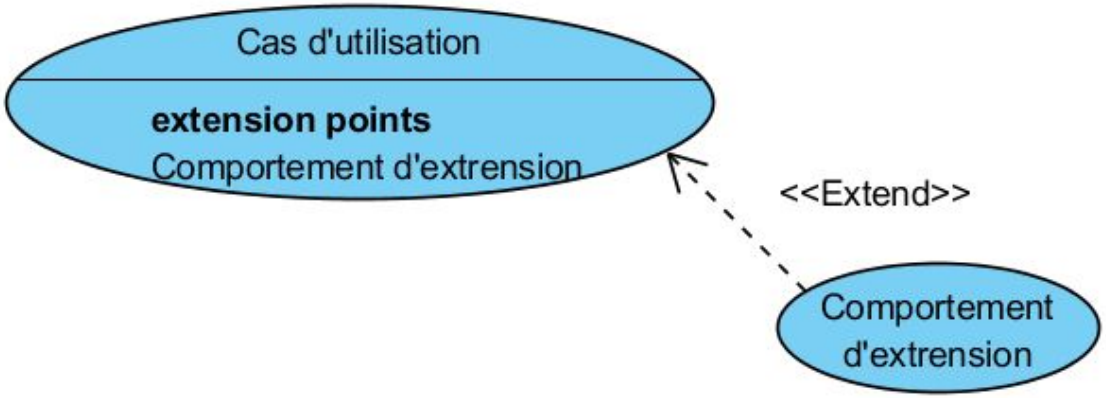
- Utilisation conseillée: **Factorisation** d'une partie de comportement commune entre plusieurs Use Case



Extend

- Permet de décrire qu'un cas d'utilisation peut contenir **conditionnellement** un comportement décrit dans le "cas d'utilisation" d'extension

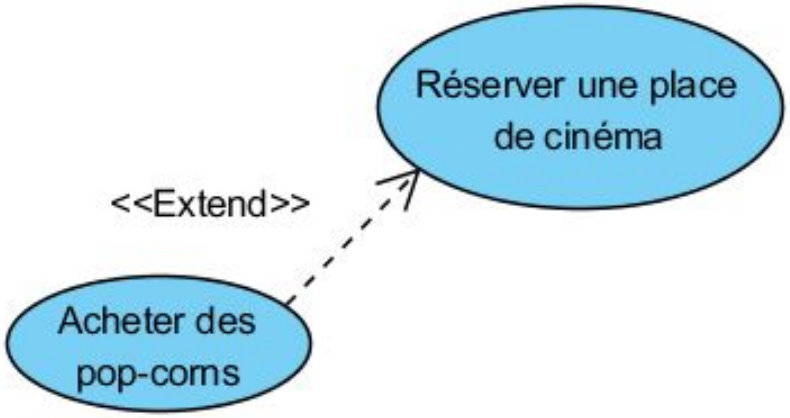
- Représentation:



Extend

Bonnes pratiques

- Utilisation conseillée: Modélisation des *scénarii alternatifs* d'un Use Case qui **étendent l'objectif initial** du Use Case

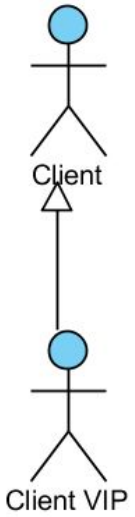


Généralisation/Spécialisation

Décrit qu'un élément (acteur ou use case) est une spécialisation d'un autre élément

- Permet de rajouter du comportement supplémentaire

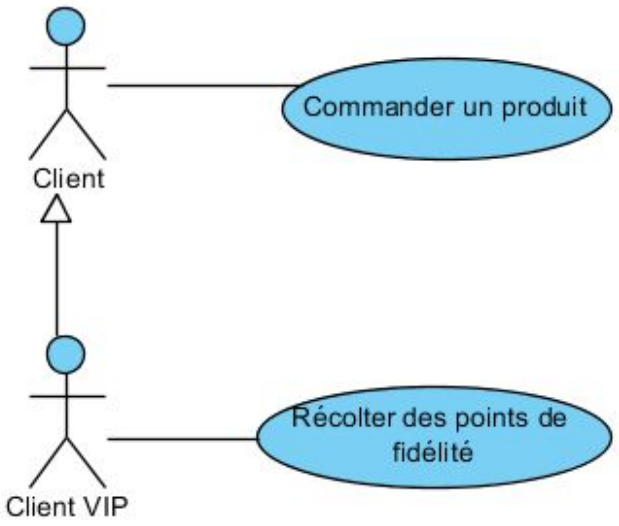
- Représentation:



Généralisation/Spécialisation

Bonne pratique:

- Utiliser les associations de généralisation/spécialisation principalement pour les acteurs



Un *Client* peut commander un produit

Un *Client VIP* peut commander un produit et récolter des points de fidélité

Généralisation/Spécialisation

Bonne pratique:

- Pour les associations de généralisation/spécialisation entre Use Cases, assurez vous que les Use Cases spécialisés respectent les Pré/Post du Use Case général

Relations

Etude de cas - Bibliothèque:

Identifier les éventuelles relations de types include, extend, généralisation

Exercices

Réaliser les exercices de cas d'utilisation du cahier d'exercices

Diagramme de Classe

CONSULTING BSTORM

Motivation

- Quels sont les différentes entités (matérielles ou non) qui vont être manipulées dans l'environnement de l'application (le domaine d'application)?
  - Les livres
  - Les auteurs
  - Les emprunts
- Quelles sont les caractéristiques de chacune des entités?
  - Les livres sont caractérisés par un titre, un nombre de pages...
- Quelles sont les relations entre chacune de ces entités?
  - Les emprunts concernent des livres
  - Les auteurs écrivent des livres

Définition

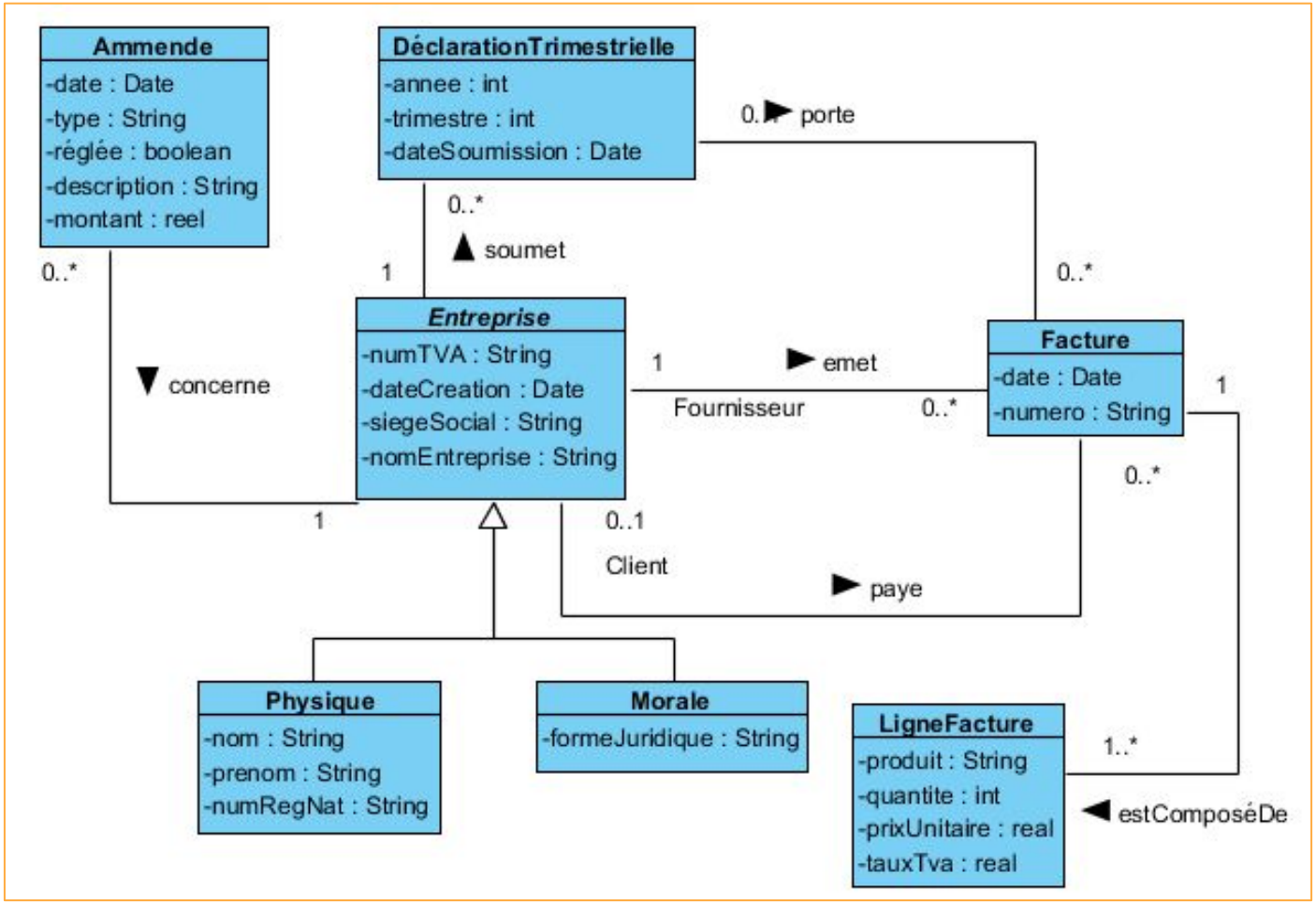
Le **diagramme de classes** est un schéma utilisé pour représenter les classes du système ainsi que les différentes relations entre celles-ci

Contexte d'utilisation

- **Analyse (Business)**
  - décrire la structure des entités manipulées dans le domaine d'application

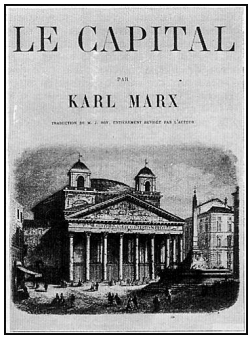
- **Conception (Système)**
  - représenter la structure d'un code orienté objet (conception)
  - représenter les classes à implémenter dans un langage de développement précis (implémentation)

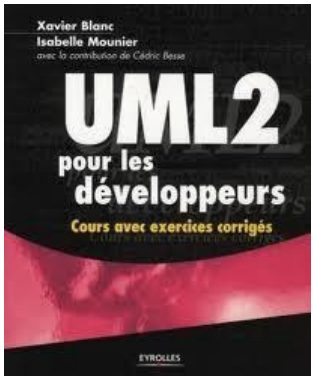
Exemple

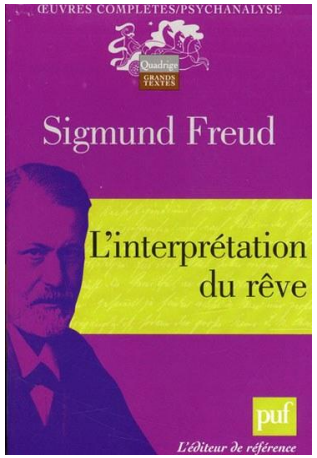

Ammende
-date : Date
-type : String
-réglée : boolean
-description : String
-montant : reel
DéclarationTrimestrielle
-annee : int
-trimestre : int
-dateSoumission : Date
0..* porte
0..*
soumet
1
0..*
0..*
Entreprise
-numTVA : String
-dateCreation : Date
-siegeSocial : String
-nomEntreprise : String
concerne
1
1
emet
Fournisseur
0..*
Facture
-date : Date
-numero : String
1
0..*
1
0..1
Client
paye
Physique
-nom : String
-prenom : String
-numRegNat : String
Morale
-formeJuridique : String
LigneFacture
-produit : String
-quantite : int
-prixUnitaire : real
-tauxTva : real
1..*
estComposéDe

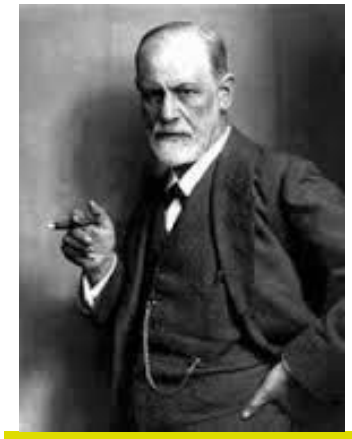
Notion Classe/Objet

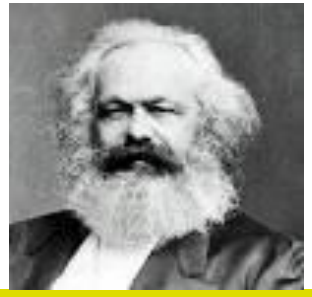
Classifier ces différents obj







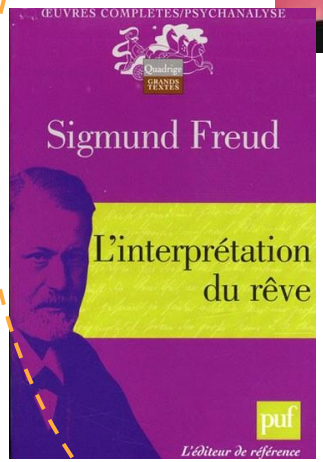


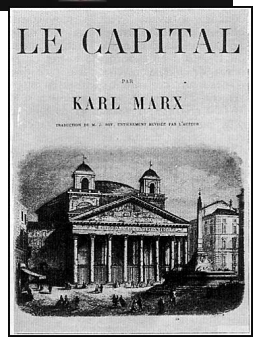


Notion Classe/Objet

Classifier ces différents objets

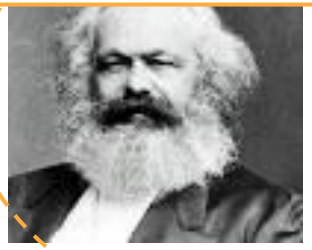
Livre

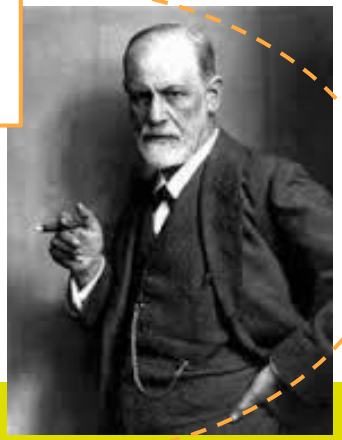




Membre de l'université

Auteur





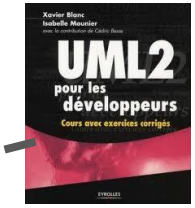
Notion Classe/Objet

CLASSE

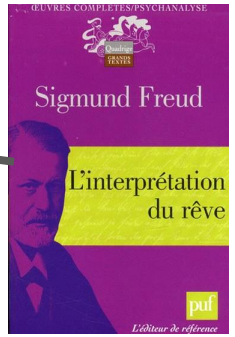
INSTANCE DE CLASSE
OBJET

Livre

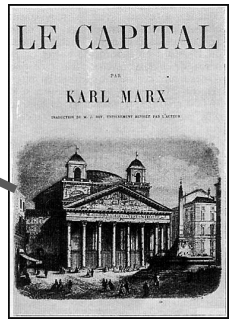
est de type



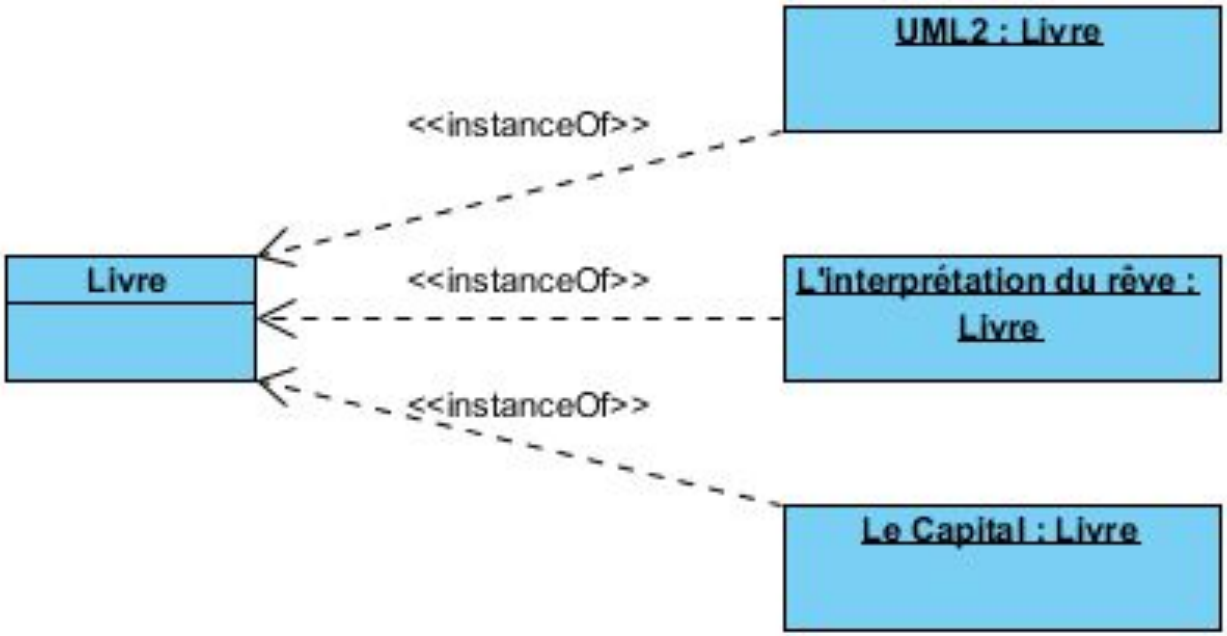
est de type



est de type



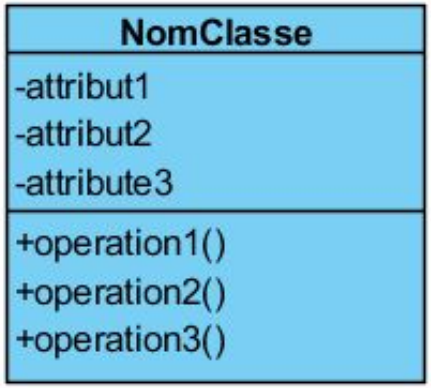
Notion Classe/Objet



Classe

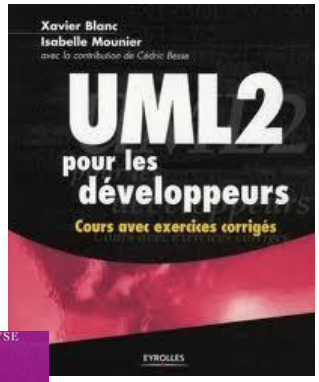
Une classe déclare des propriétés communes à un ensemble d'objets

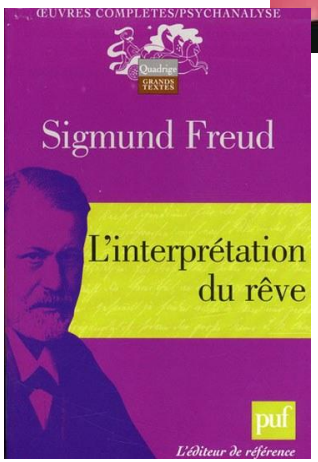
- Les attributs représentent l'état des objets
- Les opérations représentent le comportement possible des objets
- Notation:

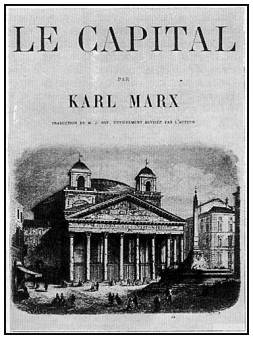


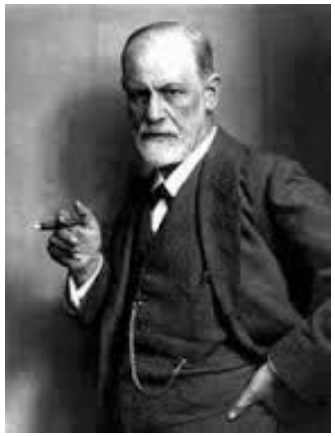
Attributs

Identifier les attributs de ces 2 classes



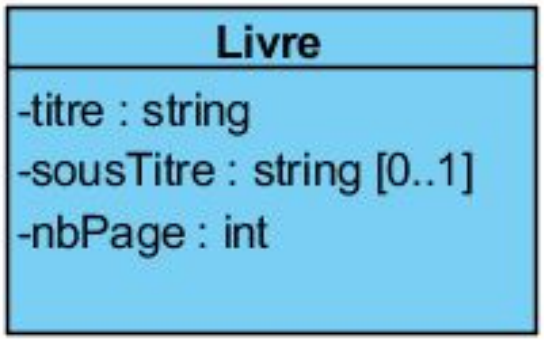






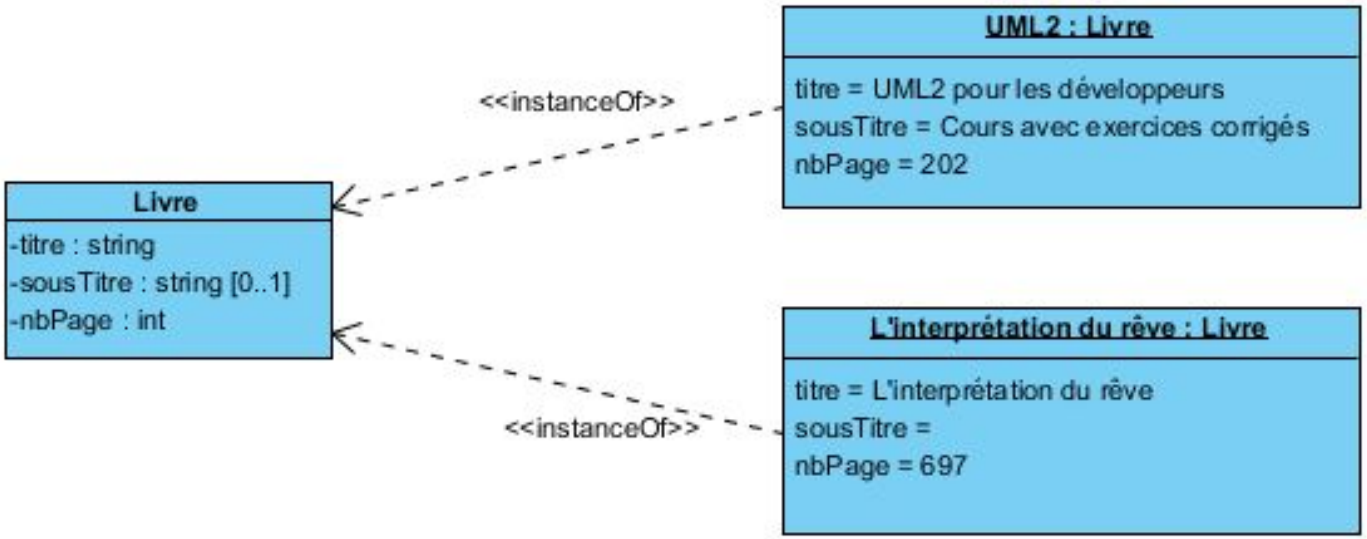
Attributs

- **Propriété structurelle** partagée par tous (?) les objets d'une classe
- Chaque objet peut avoir une **valeur** différente pour un attribut particulier
- À chaque attribut est associé un **type** qui définit un espace de valeur possible



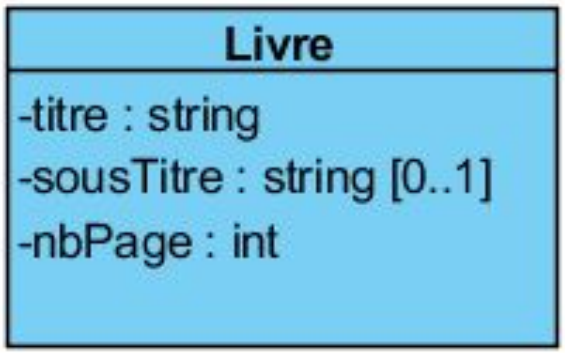
Attributs

- **Propriété structurelle** partagée par tous (?) les objets d'une classe
- Chaque objet peut avoir une **valeur** différente pour un attribut particulier
- À chaque attribut est associé un **type** qui définit un espace de valeur possible



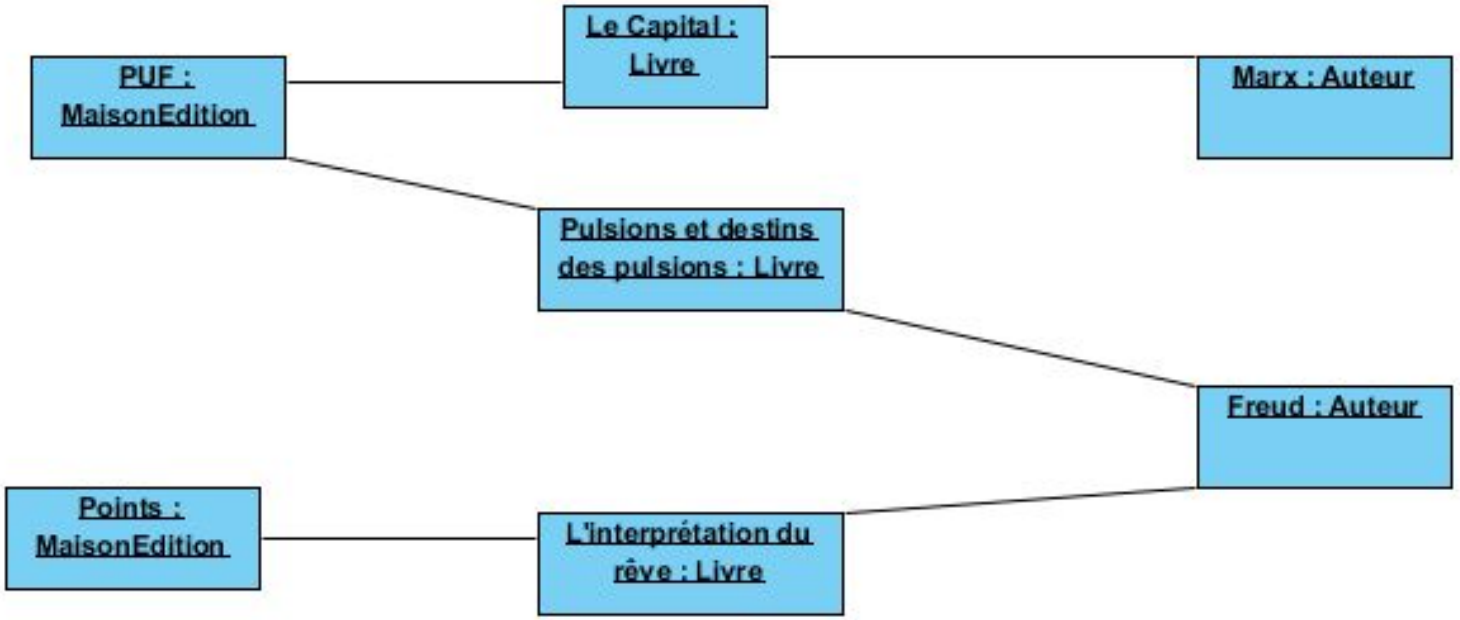
Attributs

- **Propriété structurelle** partagée par tous (?) les objets d'une classe
- Chaque objet peut avoir une **valeur** différente pour un attribut particulier
- À chaque attribut est associé un **type** qui définit un espace de valeur possible



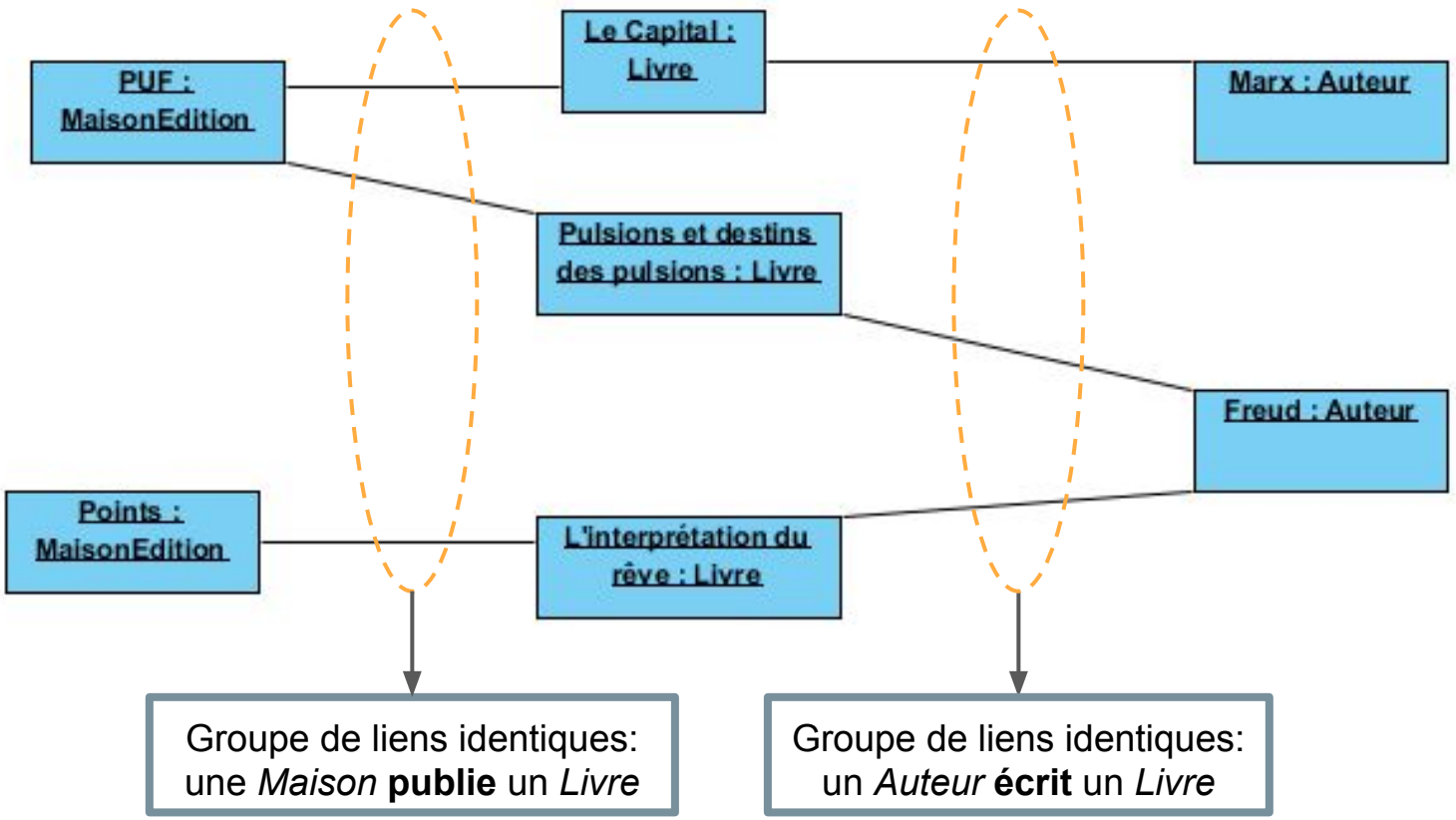
La relation d'Association

Une **association** décrit un *groupe de liens* ayant une même structure et une même sémantique



La relation d'Association

Une **association** décrit un *groupe de liens* ayant une même structure et une même sémantique



La relation d'Association

Une **association** décrit un *groupe de liens* ayant une même structure et une même sémantique

La relation d'Association

Terminologie:

- L'association est un type de **relation** (il en existe d'autres, ex.: la composition, l'aggrégation)

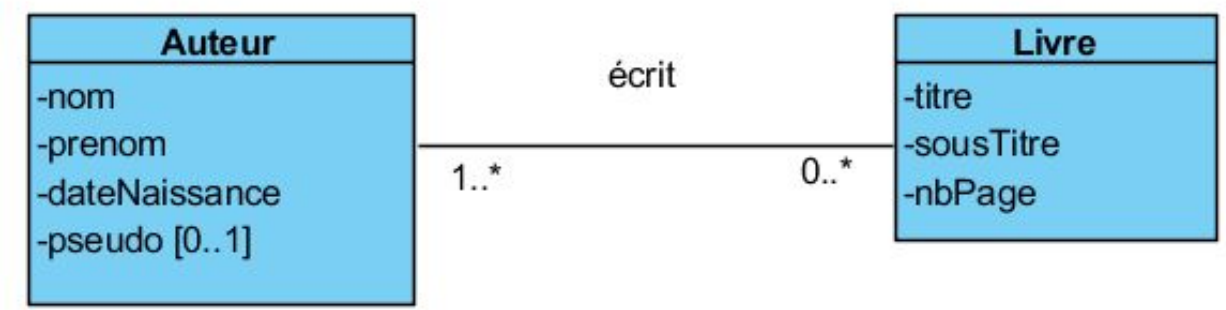
- Une instance d'association est appelé un **lien**

La relation d'Association

EXO
Propriétaire voit.

Les association sont caractérisées par:

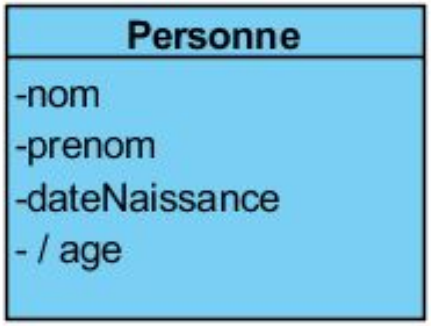
- Un nom
- Une multiplicité



Attribut dérivé

- Un **attribut dérivé** est une propriété intéressante pour l'analyse, mais redondante, car sa valeur peut être déduite d'autres informations disponibles dans le modèle

- Identifiez l'attribut dérivé dans cette classe:



Attribut dérivé

EXO
Facture

- **Durant l'analyse**, permet de ne pas faire de choix de prématuré
- **Durant la conception/implémentation**, 2 solutions:
  - Garder un attribut en mémoire et mettre sa valeur à jour adéquatement
    - ✓ Si calcul complexe et/ou mise à jour rare
  - Ne pas stocker pas de valeur redondante, mais la calculer à la demande au moyen d'une opération
    - ✓ Si calcul simple et/ou mise à jour fréquente

Attributs de classe et d'instance

EXO
Majorité

Attribut d'instance: (par défaut)

- Chaque instance peut avoir une valeur différente pour cet attribut
- Exemple: Chaque client possède son propre prénom

Attribut de classe:

- Toutes les instances partagent la même valeur pour cet attribut
- Exemple: Tous les prêts ont une durée maximale de 5 jours
- Notation: Attribut souligné

Attributs - Syntaxe et sémantique

[visibility][/]name[:type][“[”multiplicity“]”][=initial value][{property string}]

[visibility]

- `-` private (visible uniquement par les opération de la classe)
- `+` public (visible partout à l’intérieur et à l’extérieur de la classe)
- `~` package (visible au sein du package uniquement)
- `#` protected (visible au sein de la classe et par les descendants de cette classe)

[/]

- `/` attribut dont la valeur est dérivée (d’un ou plusieurs autres attributs)

Attributs - Syntaxe et sémantique

[visibility][/]name[:type][multiplicity][=initial value][{property string}]

[:type] spécification du domaine de valeur possible de l'attribut

PrimitiveType ex: boolean, string, int...

DataType ex: Date...

Enumeration

[multiplicity]

[*min*.. *max*] : eg, [0.. 1], [3.. 3]= 3, [0.. *] = *,…

Par défaut: attribut monovalué obligatoire, soit [1]

Si min = 0 : attribut ***facultatif***

Si max > 1 : attribut ***multivalué***

[=initial value] spécification de la valeur initiale de l'attribut

Attributs - Syntaxe et sémantique

[visibility][/]name[:type][multiplicity][=initial value][{property string}]

Modifier	Description
id	Property is part of the identifier for the class which owns the property.
readOnly	Property is read only (isReadOnly = true).
ordered	Property is ordered (isOrdered = true).
unique	Multi-valued property has no duplicate values (isUnique = true).
nonunique	Multi-valued property may have duplicate values (isUnique = false).
sequence (or seq)	Property is an ordered bag (isUnique = false and isOrdered = true).
union	Property is a derived union of its subsets.
redefines *property-name*	Property redefines an inherited property named ***property-name***.
subsets *property-name*	Property is a subset of the property named ***property-name***.
property-constraint	A constraint that applies to the property

Attributs

Comment distinguer un objet ou un attribut ?

- Un objet est un élément plus « important » qu'un attribut.

- Un bon critère à appliquer peut s'énoncer de la façon suivante : si l'on ne peut demander à un élément que sa valeur, il s'agit d'un simple attribut ; si l'on peut lui poser plusieurs questions, il s'agit plutôt d'un objet qui possède à son tour plusieurs attributs, ainsi que des liens avec d'autres objets.

Classes et Attributs

Etude de cas - Bibliothèque:

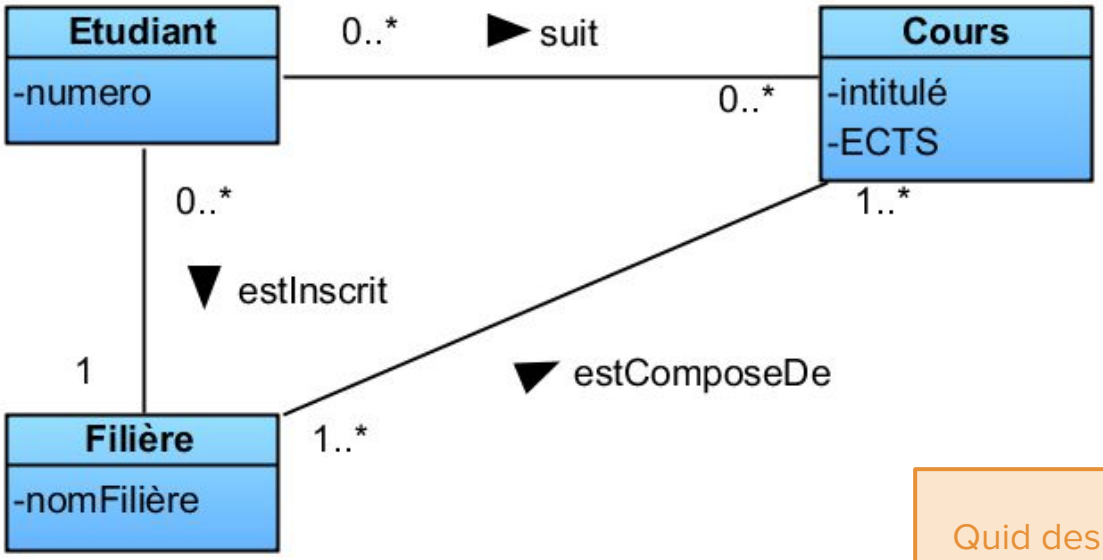
Identifier les différentes classes du domaine d'application de la bibliothèque de l'UNamur ainsi que leurs attributs respectifs

La relation d'Association

Modélisez la situation suivante:

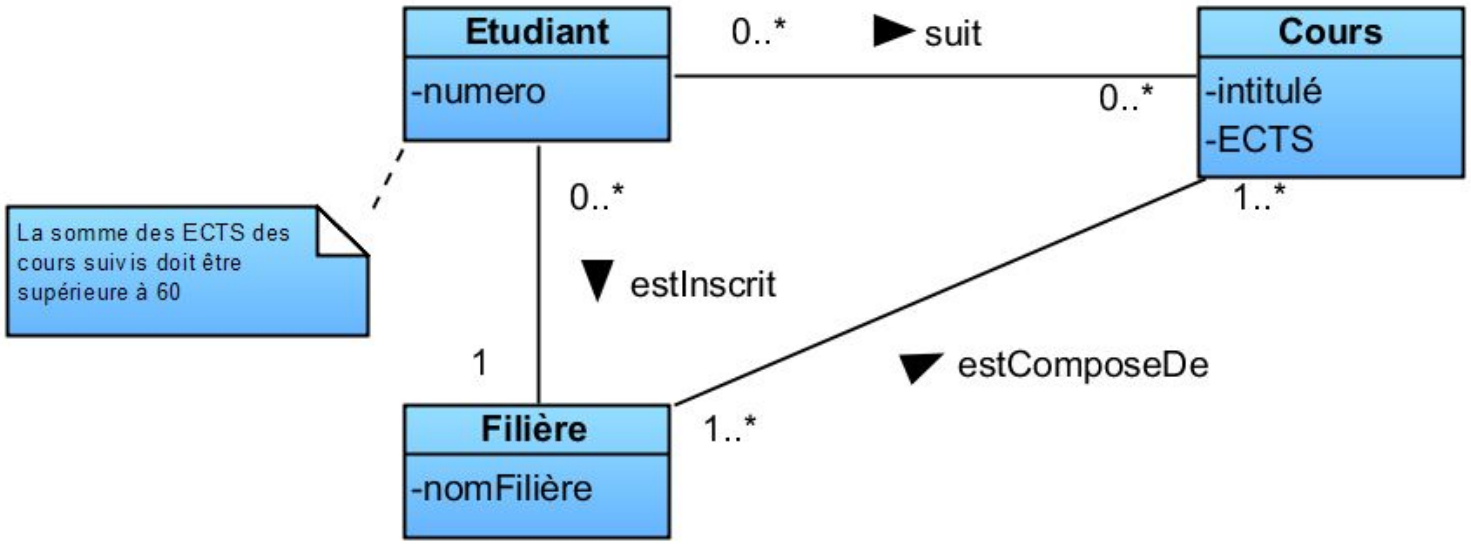
Les étudiants de l'université de Namur doivent s'inscrire à une seule filière qui propose un ensemble de cours. Chaque cours est caractérisé par un nombre de crédits ECTS (représentant la charge de travail). Au sein de ces cours, un étudiant doit en sélectionner un sous-ensemble pour l'équivalent de 60 crédits. Un cours peut se retrouver dans plusieurs filières. Les étudiants connus par un numéro ne peuvent suivre que des cours de leur filière (pas d'élève libre).

Contraintes d'intégrités sur les associations

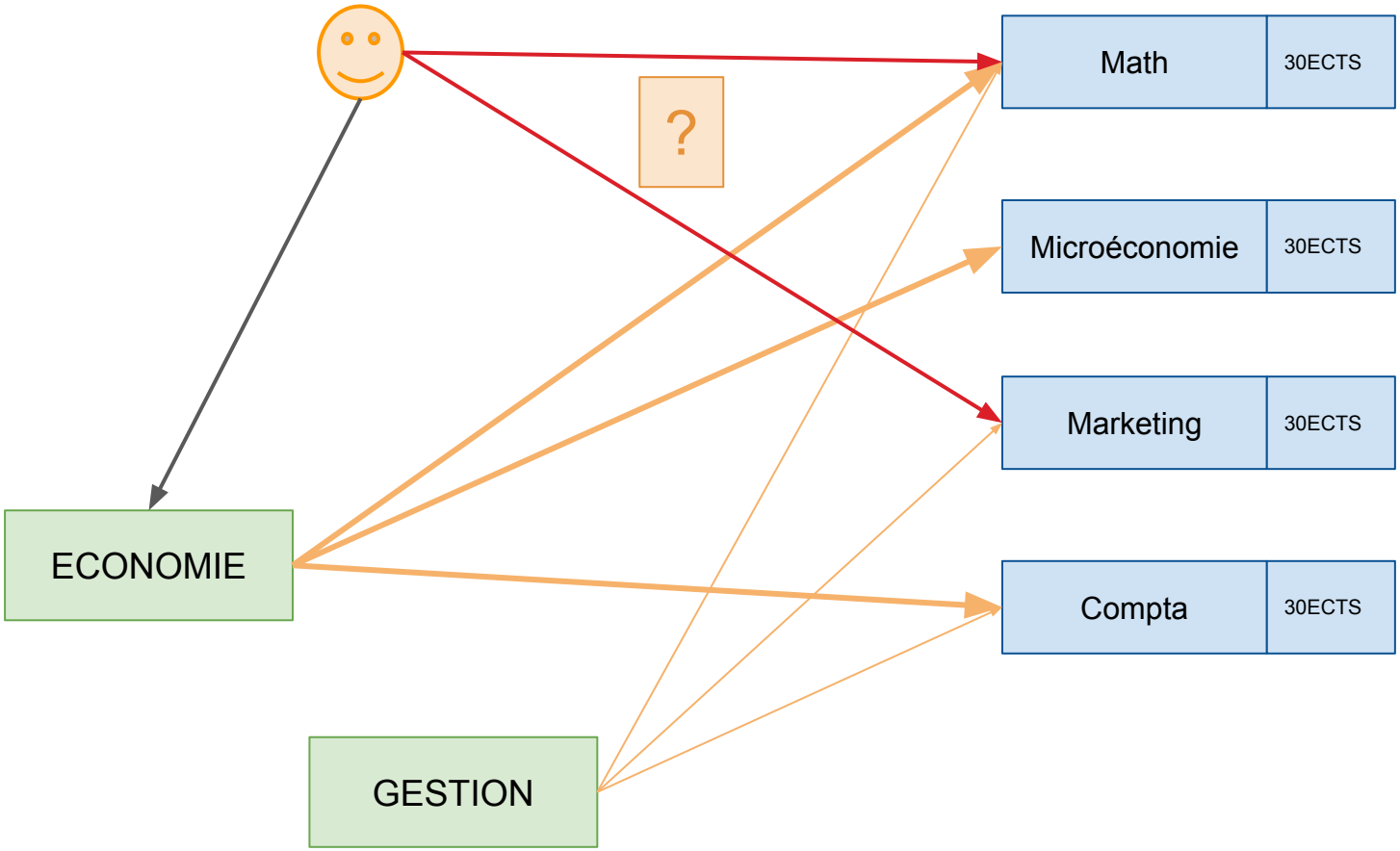


Quid des 60 ECTS minimum ?

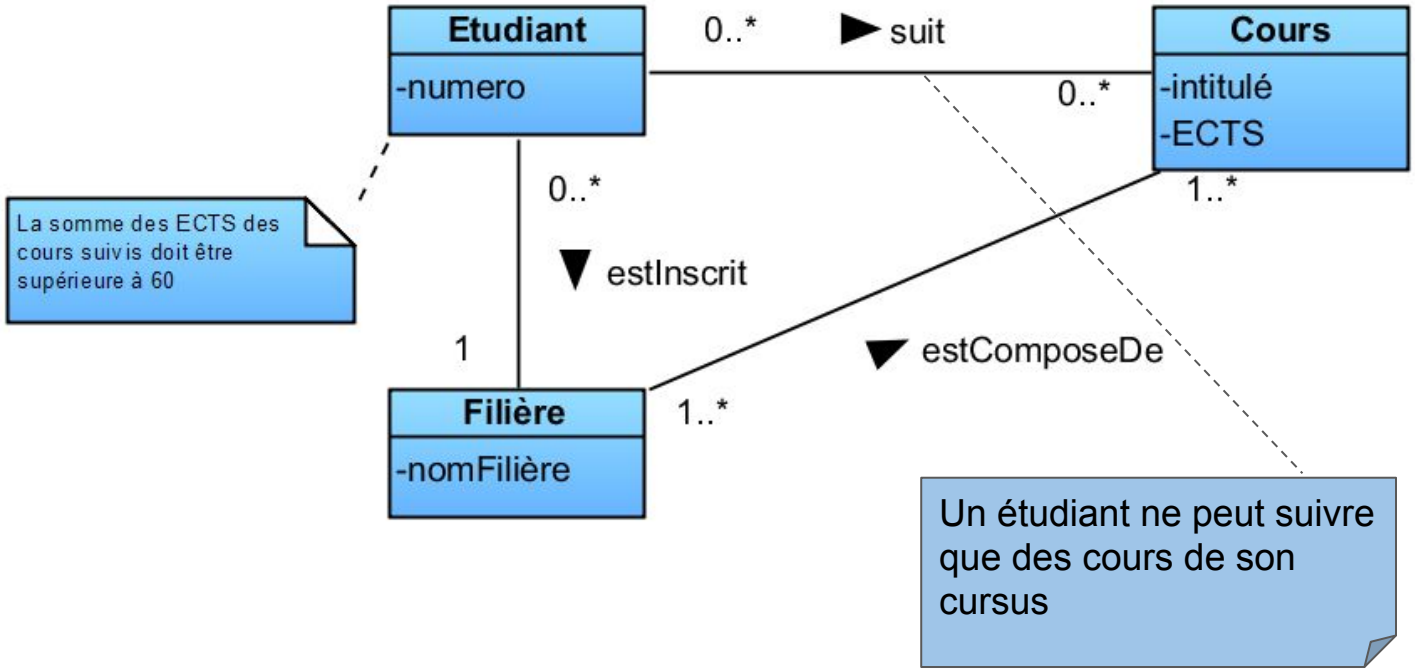
Contraintes d'intégrités sur les associations



Contraintes d'intégrités sur les associations



Contraintes d'intégrités sur les associations



La relation d'Association

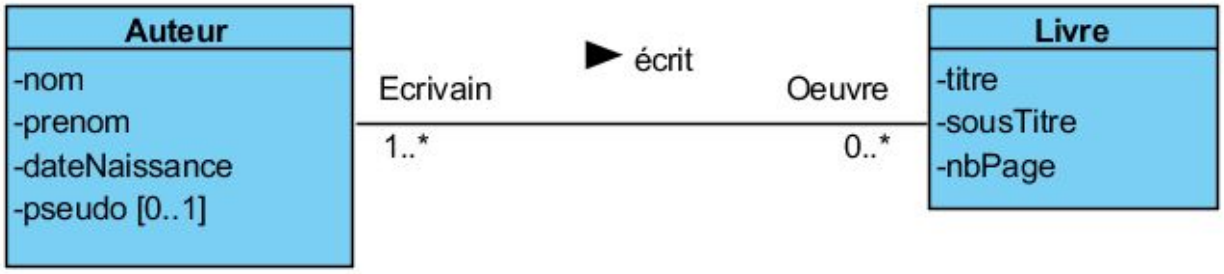
Chaque **multiplicité** peut être caractérisé par les propriétés *isUnique* et *isOrdered*

isOrdered	isUnique	Collection type
false	*true*	*Set*
true	*true*	*OrderedSet*
false	*false*	*Bag*
true	*false*	*Sequence*

La relation d'Association

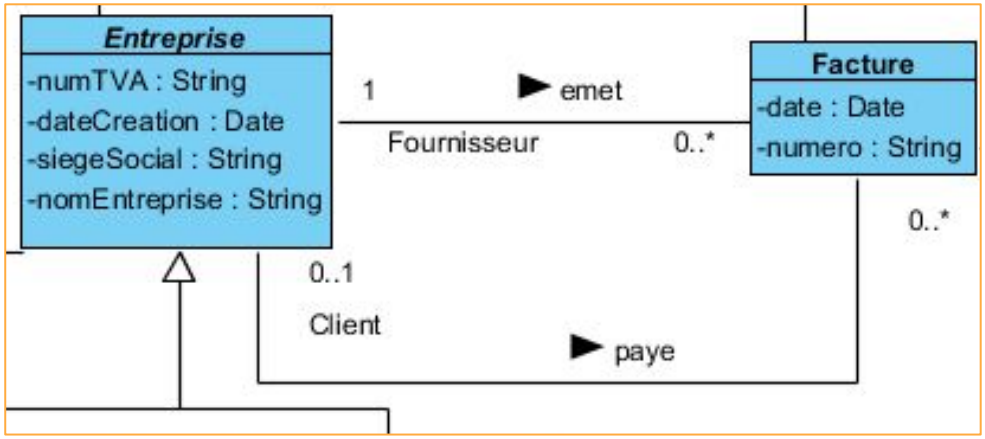
Les association *peuvent* *être* caractérisées par:

- Un sens de lecture
- Un rôle



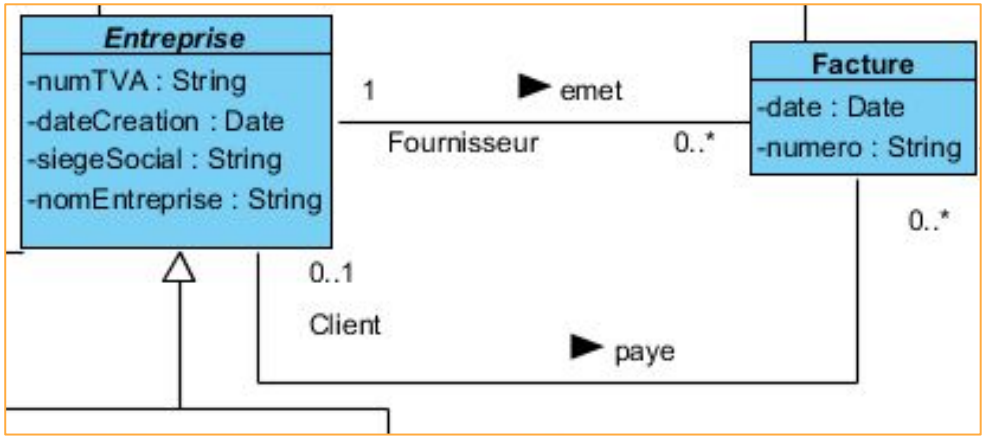
La relation d'Association

Il est nécessaire de mettre un rôle lorsqu'une classe est lié plusieurs fois à une autre même classe.



La relation d'Association

Il est nécessaire de mettre un rôle lorsqu'une classe est lié plusieurs fois à une autre même classe.

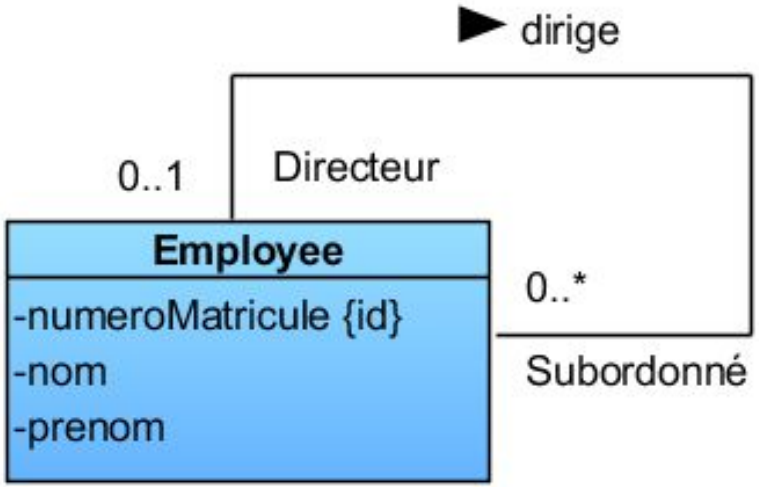



```
class abstract Entreprise {

    private String numTVA;
    private Date dateCreation;
    private String siegeSocial;
    private String nomEntreprise;

    /*
    ...

    */

}

class Facture {

    private String numero;
    private Date date;
    private Entreprise fournisseur;
    private Entreprise client;

    /*
    ...

    */
```

La relation d'Association

Association réflexive

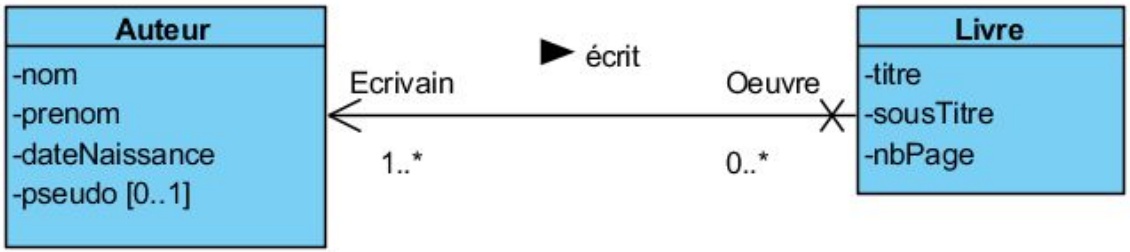
- Association dont la cible est identique à la source
- Les rôles sont obligatoires



La relation d'Association

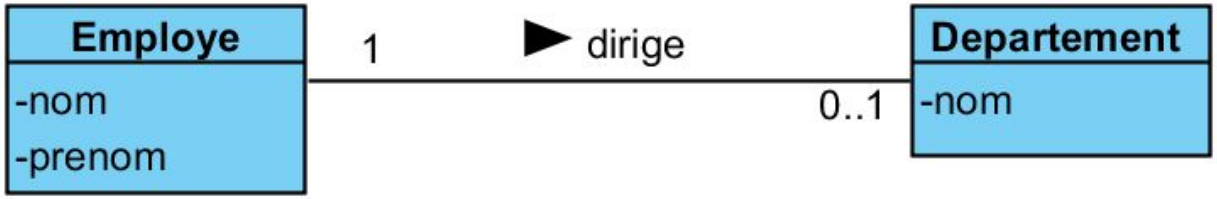
Les association *peuvent* *être* caractérisées par:

- Un sens de navigation



La relation d'Association

Sens de navigation : Comment naviguer entre les objets

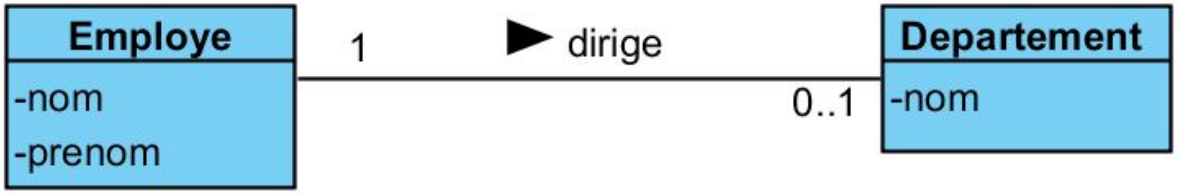



```
class Employe{


    private String nom;
    private String prenom;

    // Methods
    // ...

}
```

```
class Departement{


    private String nom;

    // Methods
    // ...

}
```

La relation d'Association

Sens de navigation : Non-specifié (Choix laissé au développeur)




```
class Employe{


    private String nom;
    private String prenom;
    //private Departement dirige;

    // Methods
    // ...

}
```

```
class Departement{


    private String nom;
    //private Employe dirige;

    // Methods
    // ...

}
```

La relation d'Association

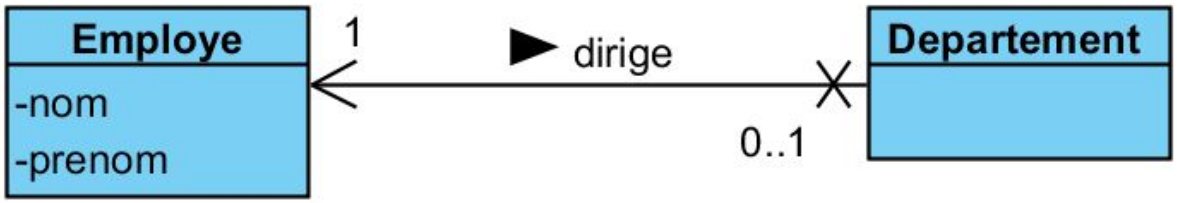
Sens de navigation : Navigable


```
class Employe{


    private String nom;
    private String prenom;
    //private Departement dirige;

    // Methods
    // ...

}
```

```
class Departement{

    private String nom;
    private Employe dirige;

    // Methods
    // ...

}
```

La relation d'Association

Sens de navigation : Non-navigable




```
class Employe{

    private String nom;
    private String prenom;


    // Methods
    // ...


}
```

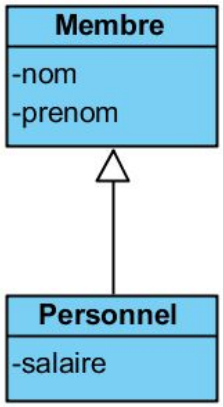
```
class Departement{

    private String nom;
    private Employe dirige;

    // Methods
    // ...

}
```

Généralisation/Spécialisation

- La relation de généralisation/spécialisation est une relation de hiérarchisation des classes d'entités

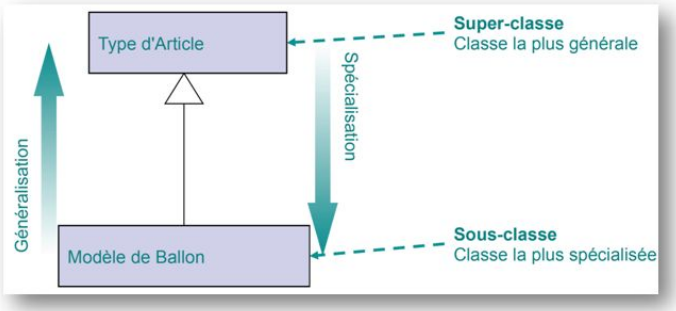
- Classification des objets dans différentes classes positionnées dans différents niveaux hiérarchiques

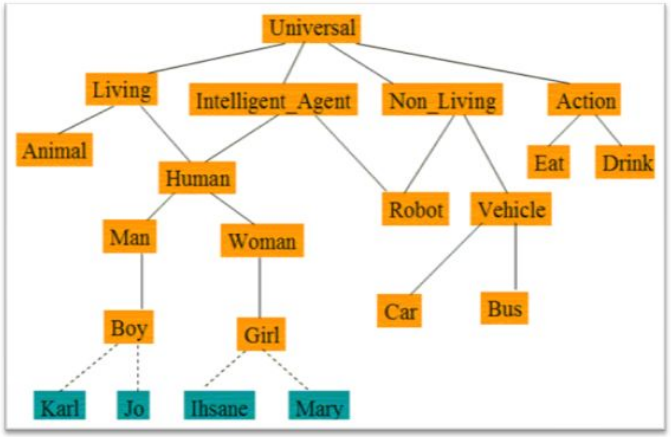
- Représentation:



Généralisation/Spécialisation

- Spécialisation: ajout de caractéristiques spécifiques aux sous-classes
- Généralisation: Factorisation de caractéristiques communes (abstraction des détails)
- Inclusion ensembliste!





Ensemble de généralisation

- {complete, disjoint}
  - Indicates the generalization set is covering and its specific Classifiers have no common instances.
- {incomplete, disjoint}
  - Indicates the generalization set is not covering and its specific Classifiers have no common instances ***(default)***.
- {complete, overlapping}
  - Indicates the generalization set is covering and its specific Classifiers do share common instances.
- {incomplete, overlapping}
  - Indicates the generalization set is not covering and its specific Classifiers do share common instances.

Généralisation/Spécialisation

Classe Abstraite:

Toutes les entités de la super classe sont obligatoirement spécialisées (la classe ne peut être instanciée)

Classe Concrète:

La classe peut être instanciée

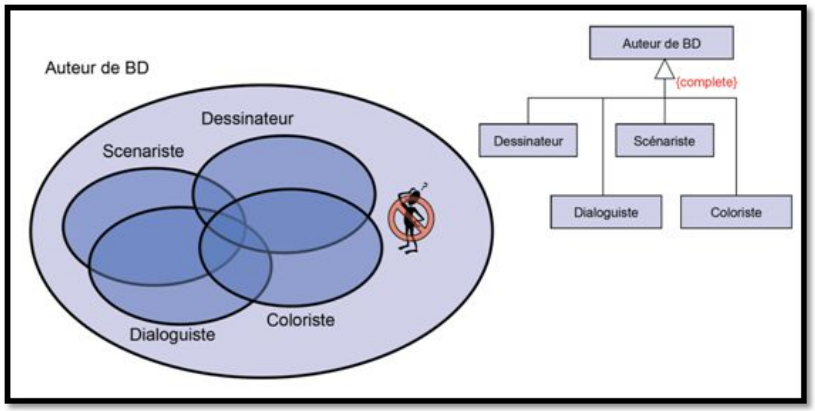
Généralisation/Spécialisation

Classe Abstraite:

Toutes les entités de la super classe sont obligatoirement spécialisées (la classe ne peut être instanciée)

Classe Concrète:

La classe peut être instanciée



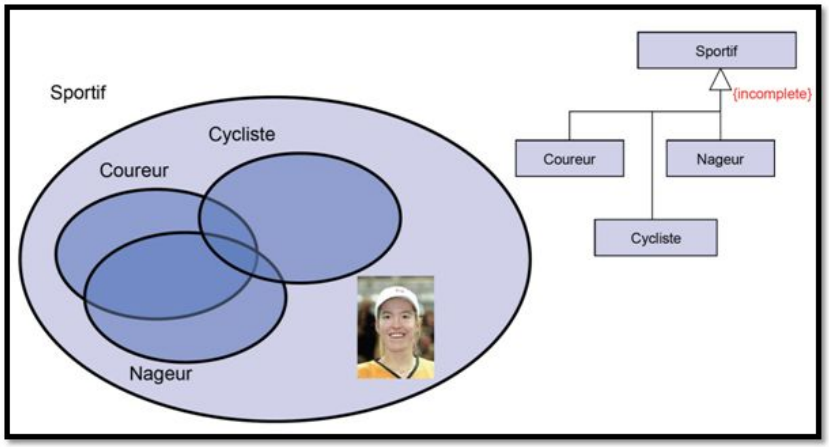
Généralisation/Spécialisation

Classe Abstraite:

Toutes les entités de la super classe sont obligatoirement spécialisées (la classe ne peut être instanciée)

Classe Concrète:

La classe peut être instanciée



Généralisation/Spécialisation

Représentation classe abstraite :

Nom de classe en italique

Bonne pratique:

N’employez la relation de généralisation que lorsque la sous-classe est conforme à 100 % aux spécifications de sa super-classe.

= *Principe de substitution de Liskov*

Généralisation/Spécialisation

Principe de substitution de Liskov:

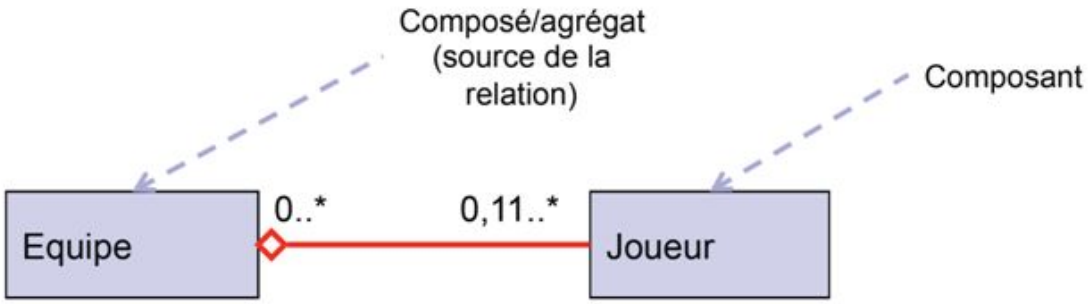
Un LivreRare est-il un Livre ?

Relation d'Agrégation/Composition

Agrégation

Indique que les instances d'une classe sont les agrégats d'instance de l'autre classe

Représentation:

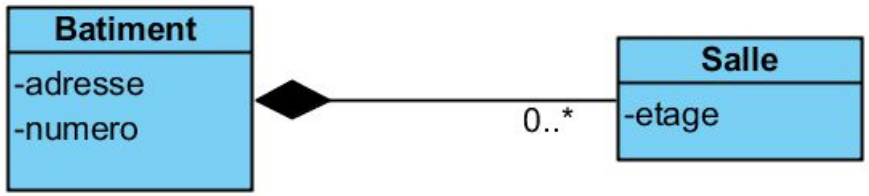


Relation d'Agrégation/Composition

Composition

- Forme forte d'agrégation
- L'existence du composant dépend strictement de l'existence du composé/agrégat
- La multiplicité du côté du composé vaut 1
- Le composant ne peut être impliqué que dans une seule composition

Représentation:



Relations

Etude de cas - Bibliothèque:

Identifier les différentes relations du domaine d'application de la bibliothèque de l'UNamur

Opérations

Représente un service/une fonction permis par les instances de la classe

Définition des responsabilités:

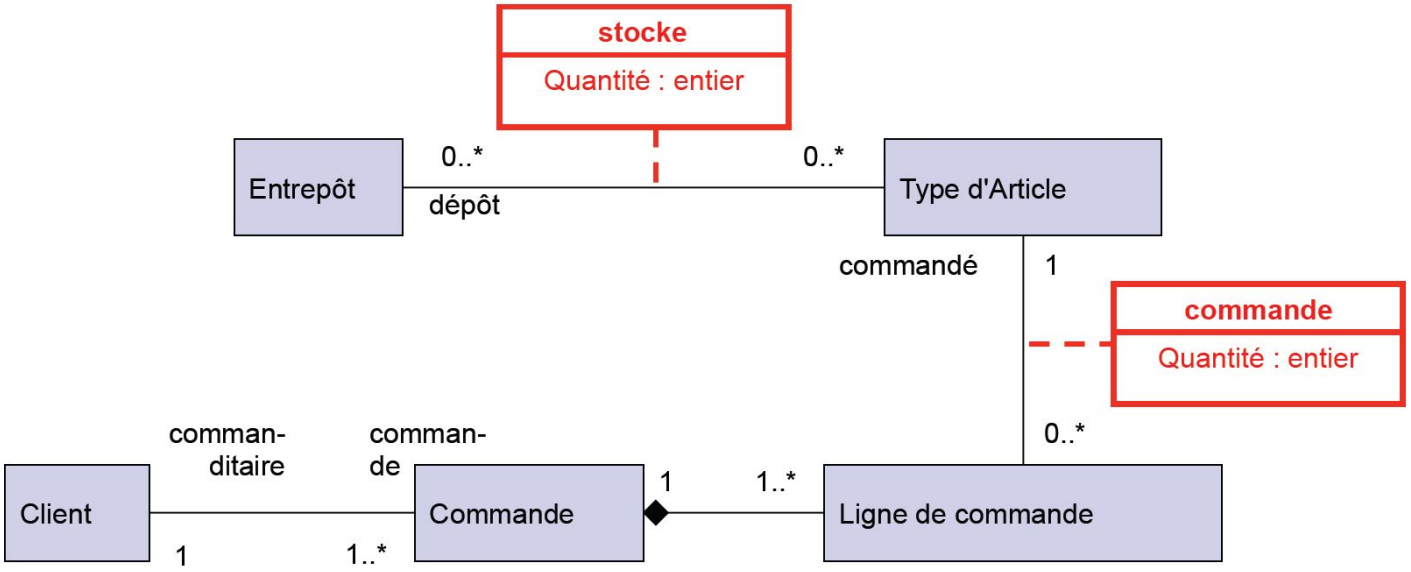
> En orienté objet, on considère que l'objet sur lequel on pourra réaliser un traitement doit le déclarer en tant qu'opération. Les autres objets qui posséderont une référence dessus pourront alors lui envoyer un message qui invoque cette opération.

Syntaxe:

[visibility] name (parameter list) [: type] [{property string}]

Classe d'association

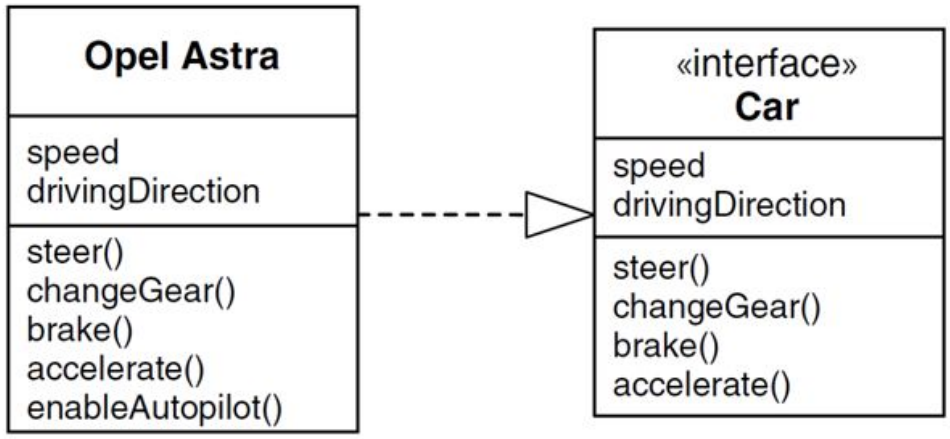
Permet de spécifier des propriétés d'une association



Interface

Ensemble des attributs et des méthodes publiques que des classes peuvent s'engager à fournir ou à exiger vis-à-vis de l'extérieur

= Contrat qu'une classe s'engage à respecter



Conventions de nommage

- Les noms des classes qui commencent par une majuscule et peuvent contenir ensuite plusieurs mots concaténés, commençant par une majuscule (**MaClasse**)

- Les noms des attributs, des rôles, des associations et des opérations commencent toujours par une minuscule (**monAttribut**)

- Il est préférable de ne pas utiliser d'accents ni de caractères spéciaux

Exercices

Réaliser les exercices de diagramme de classe du cahier d'exercices

Diagramme d’activité

CONSULTING BSTORM

Motivation

- Quelles sont les actions à entreprendre afin de louer un livre?
  - S'inscrire
  - Réserver un livre
  - Rendre un livre
  - (Déjà identifiée avec le diagramme de UC)
- Quelle est la séquence définie entre ces actions?
  - S'inscrire ⇒ Réserver un livre ⇒ Rendre un livre
- Quelles sont les conditions à chaque séquence?
  - Que faire si ISBN est illisible?

Définition

Un diagramme d'activité permet de modéliser le **comportement du système**, dont la séquence des actions et leurs conditions d'exécution

Les actions sont les unités de base du comportement du système

Contexte d'utilisation

- **Analyse (Business)**
  - Modélisation du workflow d'un Use Case
  - Modélisation du workflow entre plusieurs Use Case
  - Modélisation d'un Business Process

Action

Une action représente une étape unique au sein d'une activité, c-à-d. qu'elle ne n'est plus décomposée au sein de l'activité

- Constitue l'unité fondamentale de fonctionnalité exécutable dans une activité
- Peut représenter une transformation ou un calcul quelconque dans le système modélisé

- Peuvent être liées à des opérations

Notation:

Activité

Une activité spécifie la coordination de l'exécution de comportements subordonnés. Il modélise des flux de contrôle et de données

- Une activité est un comportement constitué de sous-comportement (sous-activités, actions…)

Notation:

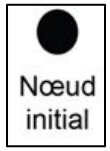
Différence entre Action et Activité

L’activité définit un comportement qui peut être réutilisé à différents endroits tandis qu’une action n’est utilisée qu’à un seul endroit dans une activité bien particulière.

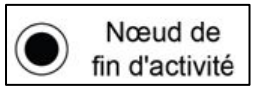
Noeuds de début et de fin d'activité

Nœud initial:

indique le point de départ (unique) d'une activité



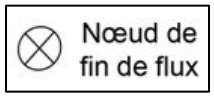
Nœud de fin d'activité :



indique la fin d'une activité, c.à.d. la fin de tous ses flux

Nœud de fin de flux:

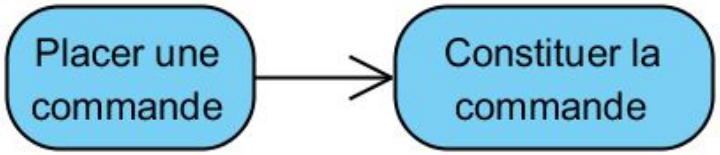
indique la fin d'un flux (d'autres flux concurrents peuvent subsister et ainsi continuer l'activité)



Flux de contrôle

Un flux de contrôle est un arc spécifiant de démarrer une activité dès que la précédente est terminée

Notation:

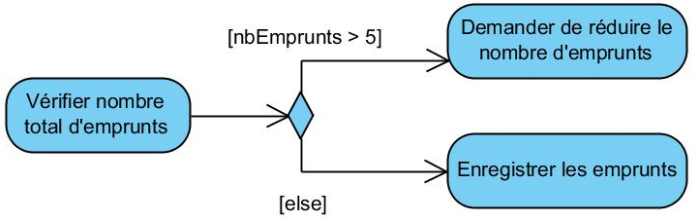


Noeud de décision

Un noeud de décision permet de rediriger les tokens entrant au travers d'un unique arc sortant.

L'arc sortant choisi dépendra du résultat de l'évaluation de la garde (la condition)

Nota



1 seul flux entrant

Plusieurs flux sortant

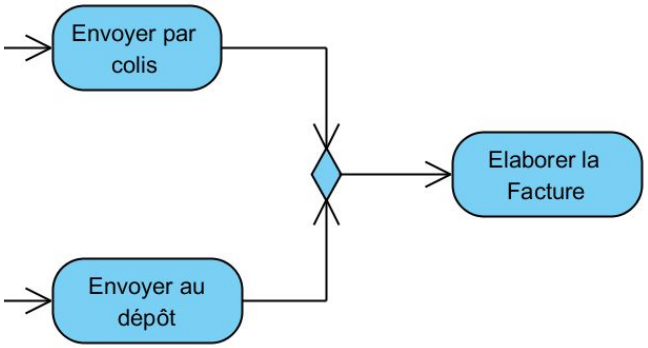
1 token entrant
=
1 token sortant

Noeud de fusion

Noeud de contrôle permettant de réunir différents flux.

Pour un token entrant, il y a toujours un seul token sortant

Notation:



Plusieurs flux entrant

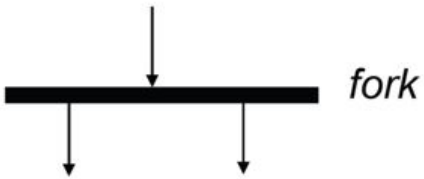
1 seul flux sortant

1 token entrant
=
1 token sortant

Noeud fourche (Fork Node)

Noeud de contrôle permettant de séparer un flux entrant en différent flux sortant **parallèle**

Pour un token entrant, il y a autant de token sortant que de flux sortant



1 seul flux entrant

Plusieurs flux sortant

1 token entrant
=
Tous les tokens sortant

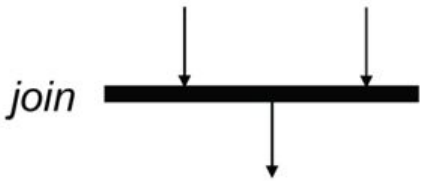
Notation:

Noeud de jointure (Join Node)

Noeud de contrôle permettant de réunir différents flux.

Pour un token entrant, il y a toujours un seul token sortant

Notation:



Plusieurs flux entrant

1 seul flux sortant

Tous les tokens entrant
=
1 token sortant

Send Signal Action

Action qui crée un signal transmit à un objet

- Peut déclencher:
  - une transition d'une machine à état,
  - l'exécution d'une activité
- Action asynchrone:
  - l'exécution de l'activité continue immédiatement après l'envoi du signal (aucune réponse n'est attendue par cette action)

Notation:

Accept Event Action

Action qui attend l'occurrence d'un événement particulier

- Peut être lié à plusieurs évènements
- Peut ne pas avoir le flux entrant

Notation:

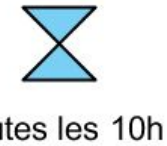
Accept Time Event Action

Action qui attend l'occurrence d'un événement temporel particulier

Deux possibilités d'exprimer un événement temporel:

- Temps absolu
- Temps relatif

Notation:



Sémantique d'un Accept Event Action

Sans flux entrant:

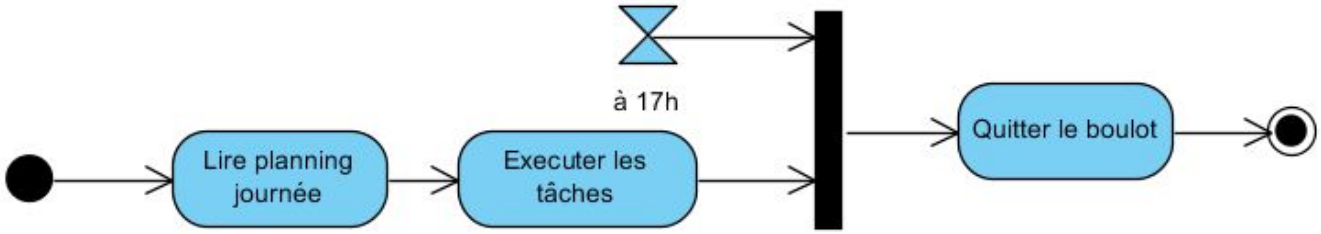
Génère un token à chaque fois qu'un événement est accepté

Avec flux entrant:

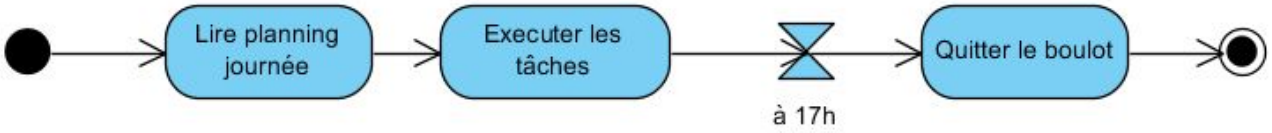
Ne peut accepter un évènement que lorsqu'un token est en attente de l'action

Sémantique d'un Accept Event Action

Sans flux entrant:

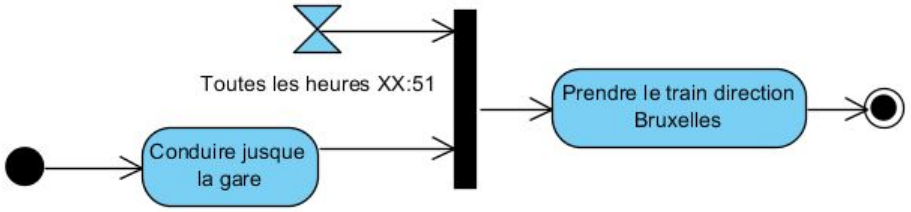


Avec flux entrant:

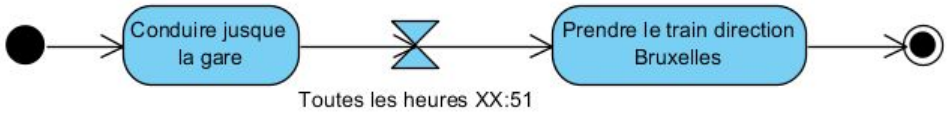


Sémantique d'un Accept Event Action

Sans flux entrant:

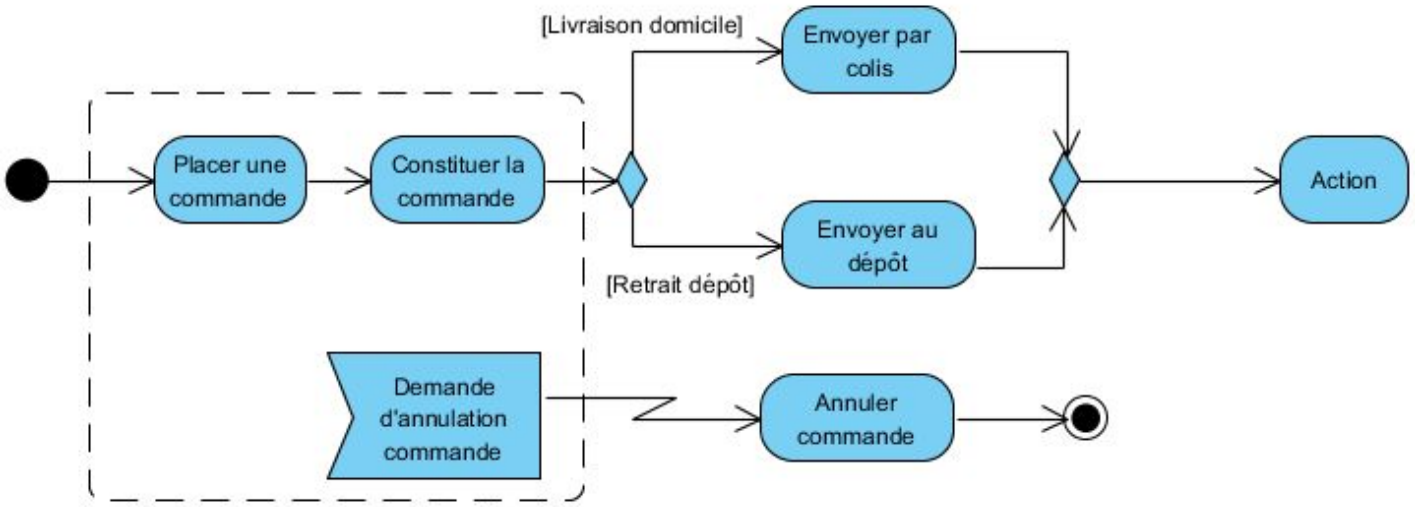


Avec flux entrant:



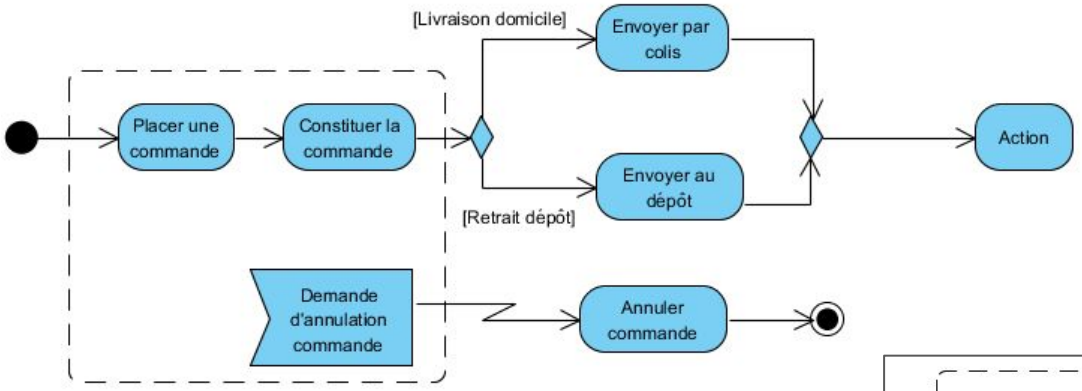
Zone Interruptible

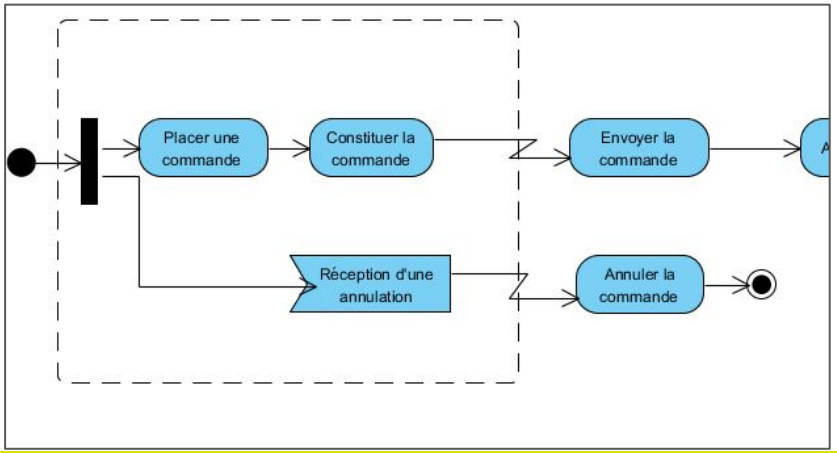
Lorsqu'un token traverse une Zone Interruptible via un arc interruptible, tous les autres tokens présents dans la région sont terminés



Zone Interruptible

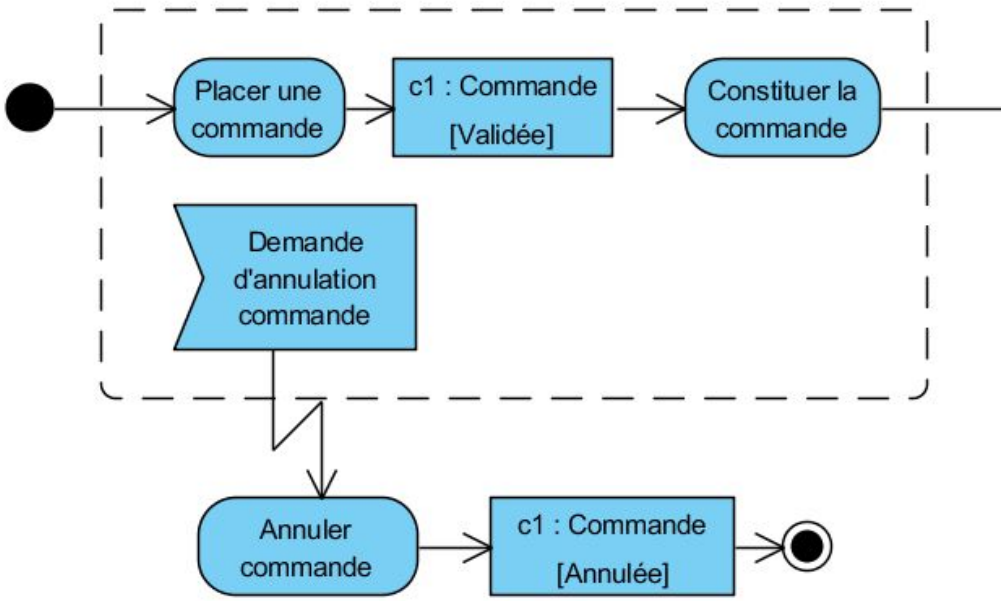
Lorsqu'un token traverse une Zone Interruptible via un arc interruptible, tous les autres tokens présents dans la région sont terminés

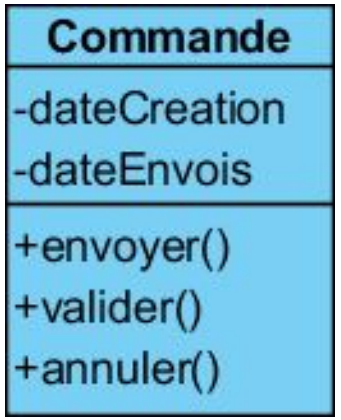




Noeud Objet

Permet de spécifier l'échange d'objet entre activités

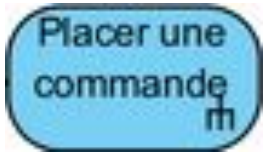


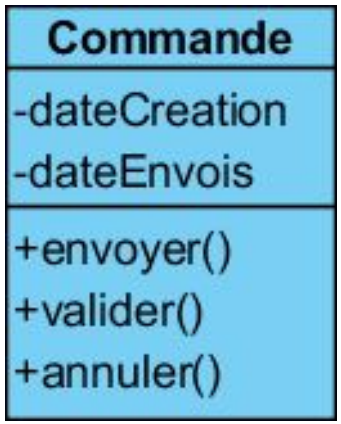


Notions avancées

Types d'action:

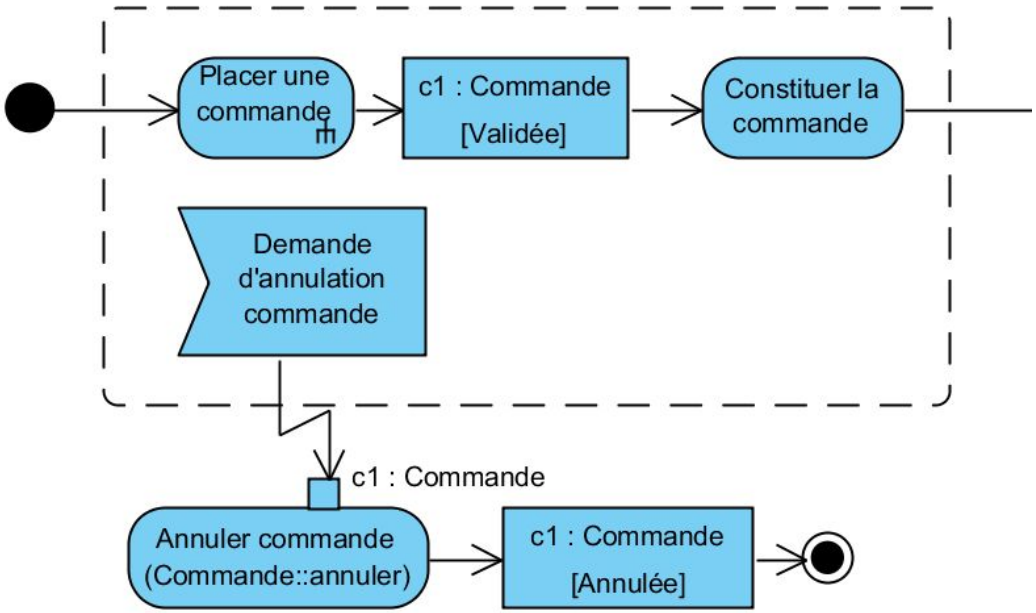
- **Call Behavior Action**
  - Référence vers un comportement (ex.: activité, machine à état)

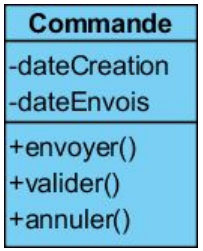




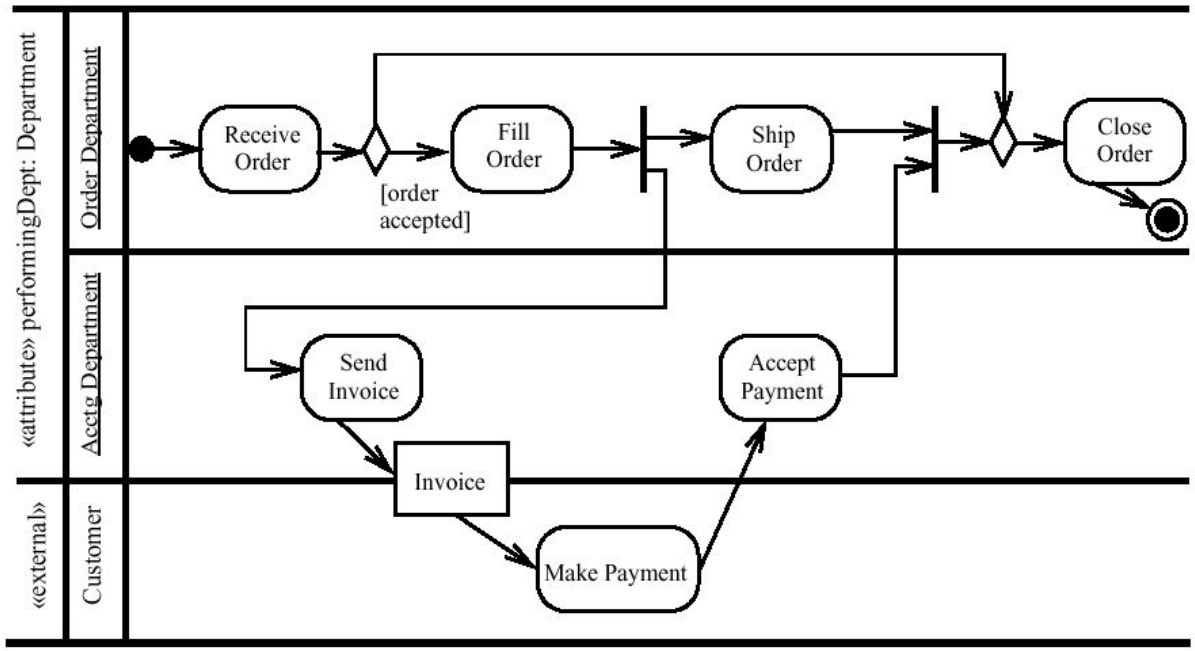
- **Call Operation Action**
  - Référence vers une opération d'un objet

Notions avancées





Swimlanes


«attribute» performingDept: Department
Order Department
Acctg Department
«external»
Customer
Receive Order
[order accepted]
Fill Order
Ship Order
Close Order
Send Invoice
Accept Payment
Invoice
Make Payment

Swimlanes

<<external>>

Permet de spécifier qu'il s'agit d'un acteur externe qui n'interagit pas avec le système

Diagramme de Séquence

CONSULTING BSTORM

Motivation

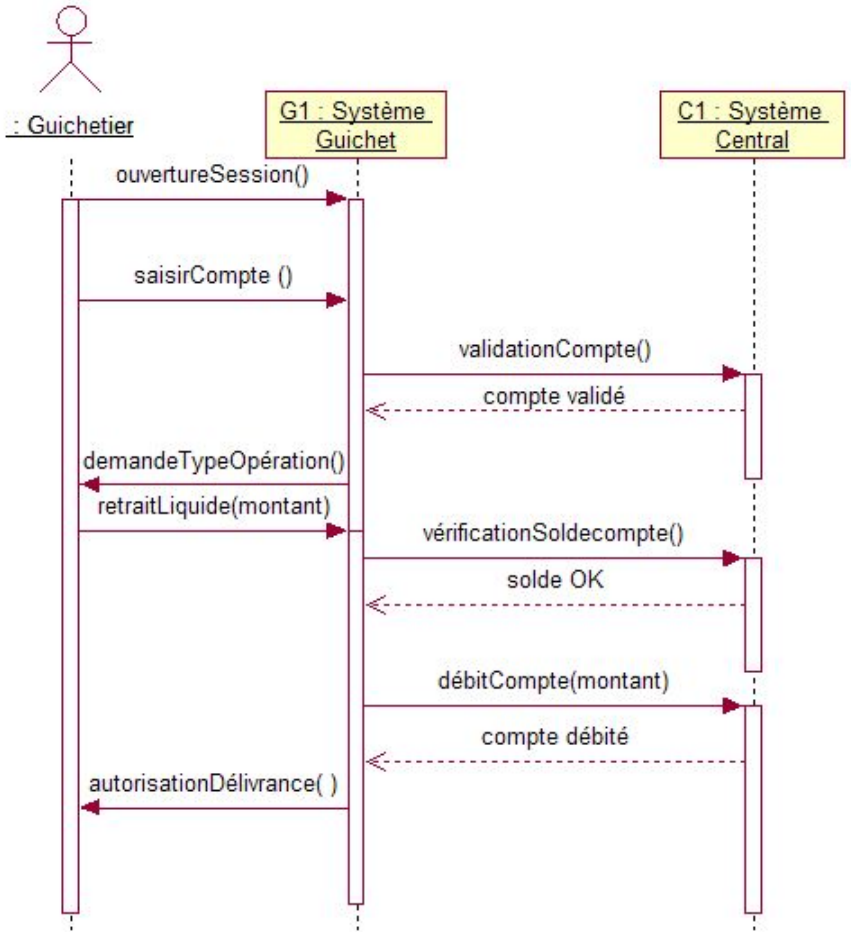
Quelle est la séquence de message échangées entre un emprunteur, un emprunt et des livres lors de la création d'un emprunt ?

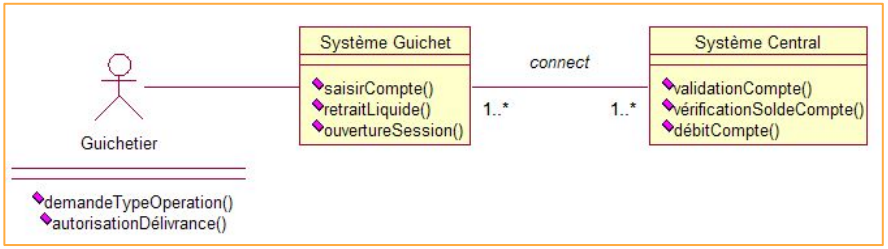
Definition

Un diagramme de séquence décrit une série de messages échangés entre un ensemble d'objets (dans une limite temporelle)

- Les objets sont impliqués dans les échanges (pas les classes)
- Permet de définir la chronologie des échanges

Overview


: Guichetier
G1 : Système Guichet
C1 : Système Central
ouvertureSession()
saisirCompte ()
validationCompte()
compte validé
demandeTypeOpération()
retraitLiquide(montant)
vérificationSoldecompte()
solde OK
débitCompte(montant)
compte débité
autorisationDélivrance( )


Guichetier
demandeTypeOperation()
autorisationDélivrance()
Système Guichet
saisirCompte()
retraitLiquide()
ouvertureSession()
connect
1..*
1..*
Système Central
validationCompte()
vérificationSoldeCompte()
débitCompte()

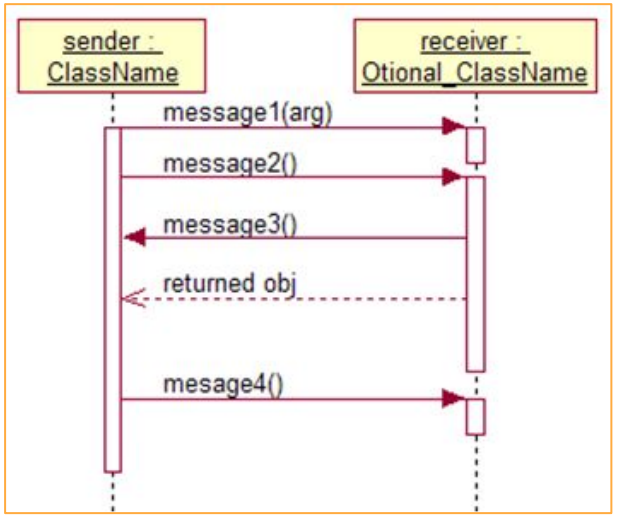
Life line

Ensemble des éléments relatifs à un participant

Implique un objet/acteurPas de Multi-objets!

Occurrences d'exécution

- Rectangles sur la lifeline
- Représentent l'exécution d'une opération de l'objet
- Début/fin définis par des événements (envoi et réception d'un message)



Rôle

Un **rôle** est attaché à chaque lifeline (*nom:type*)

- Un rôle est un objet ou un acteur
  - Ne pas modéliser les interactions directes entre acteurs (le SI n'y est pas impliqué!)
- Un acteur est souvent l'instigateur d'une séquence
- Un rôle peut-être nommé ou non-nommé

Messages

Communication entre deux rôles dans le but de transmettre des informations

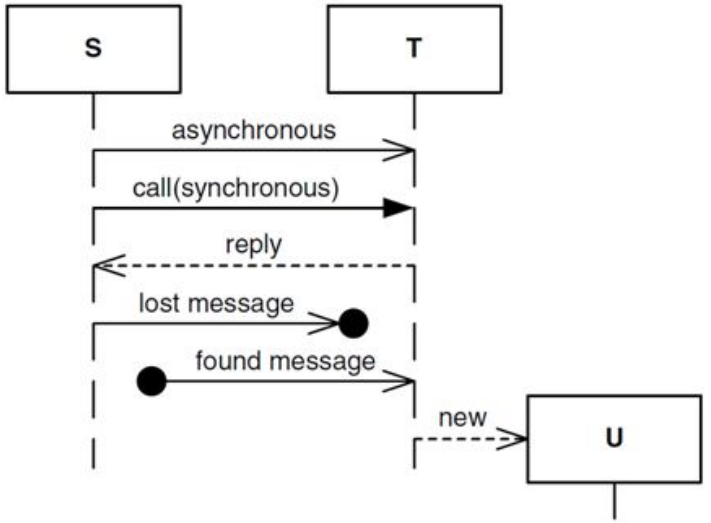
- Représente les **évènements** du diagramme de séquence
- La flèche représentant un message est labélisée avec le nom du message échangé ainsi que ses paramètres
  - Existence de messages réflexifs
  - Indication de la séquence à l'aide de nombres
- L'exécution d'une occurrence (rectangle fin) représente la participation active d'un objet/acteur dans une séquence

Type de Message

Message synchrone: effet bloquant pour l'émetteur (attente de la réponse)

Message asynchrone: l'émetteur peut continuer ses tâches
(souvent, pas de réponses prévues)

Messages perdus: l'émetteur est connu mais pas le récepteur



Messages trouvés: le récepteur est connu mais pas l'émetteur

Message création/destruction: Conduit à la création/destruction d'un objet (nouvelle/destruction d'une lifeline à cet endroit)

Syntaxe: [attribute =] name [(arguments)] [:return value]

Fragment

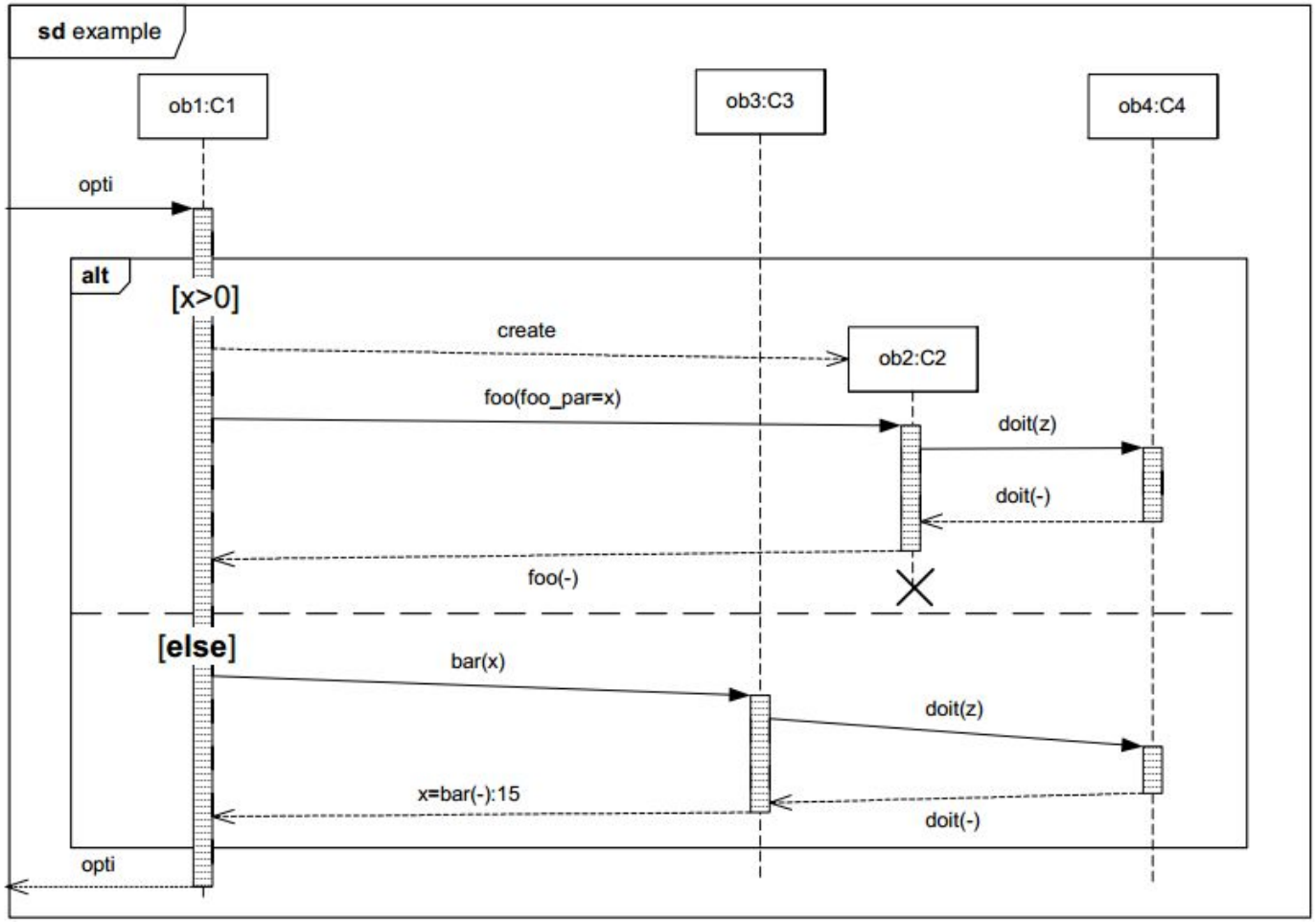

sd example
ob1:C1
ob3:C3
ob4:C4
opti
alt
[x>0]
create
ob2:C2
foo(foo_par=x)
doit(z)
doit(-)
foo(-)
[else]
bar(x)
doit(z)
x=bar(-):15
doit(-)
opti

Diagramme d’état

CONSULTING BSTORM

Motivation

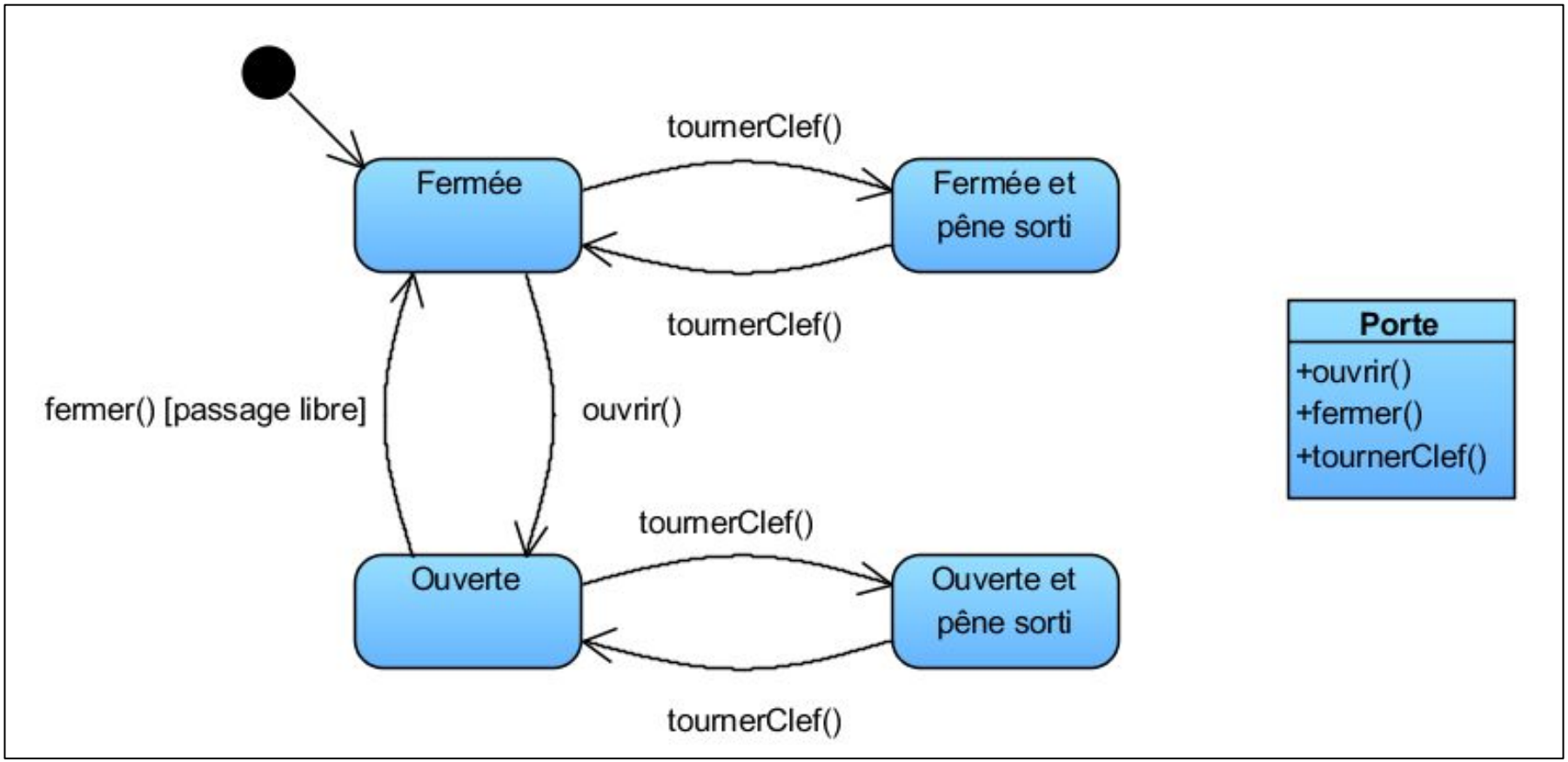
Quel est le cycle de vie d'un livre, quels sont les différents états par lequel il peut passer ?

- Réservé
- Emprunté
- …

Quelles sont les évènements qui déclenchent le passage d'un état à l'autre ?

- Restitution par le client
- ...

Exemple


tournerClef()
Fermée
Fermée et
pêne sorti
tournerClef()
fermer() [passage libre]
ouvrir()
tournerClef()
Ouverte
Ouverte et
pêne sorti
tournerClef()
Porte
+ouvrir()
+fermer()
+tournerClef()

Etat

Un état modélise une situation durant laquelle un ensemble de conditions ne varient pas.

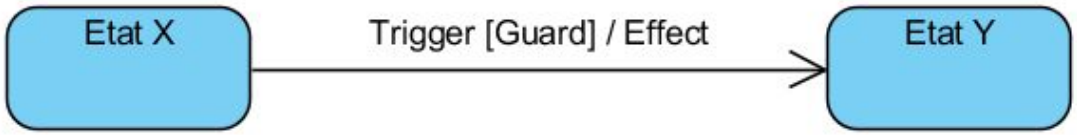
- Invariant statique
- Invariant dynamique
  - Ex.: dans un état "actif" une minuterie est constamment incrémentée

Notation:

Transition

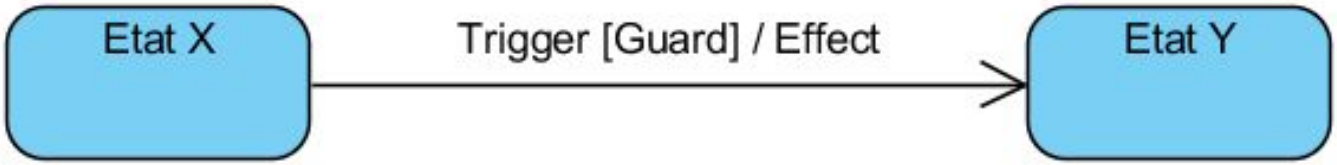
Relation dirigée (=avec un sens) entre un état source et un état cible

Notation:



Transition

- Trigger [0..*]
  - Spécifie le ou les évènements qui déclenchent la transition
- Guard [0..1]
  - Spécifie la condition qui rend possible (*enable*) la transition
- Effect [0..1]
  - Spécifie un comportement à effectuer lorsque la transition est déclenchée



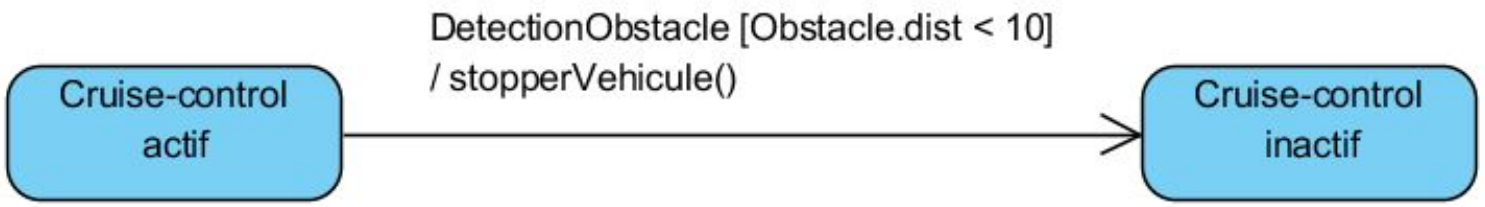
Transition

Types de trigger

- Un **Call Event** représente un appel à une opération particulière d'un objet
- Un **Change Event** modélise le fait qu'une condition devient vraie
- Un **Signal Event** représente la réception d'un message asynchrone
- Un **Time Event** spécifie un moment particulier dans le temps

Transition

Exemple:



Pseudo-Etat Initial

Permet de spécifier l'état initial par défaut d'une machine à état ou d'une région d'un état

Notation: 

Etat Final

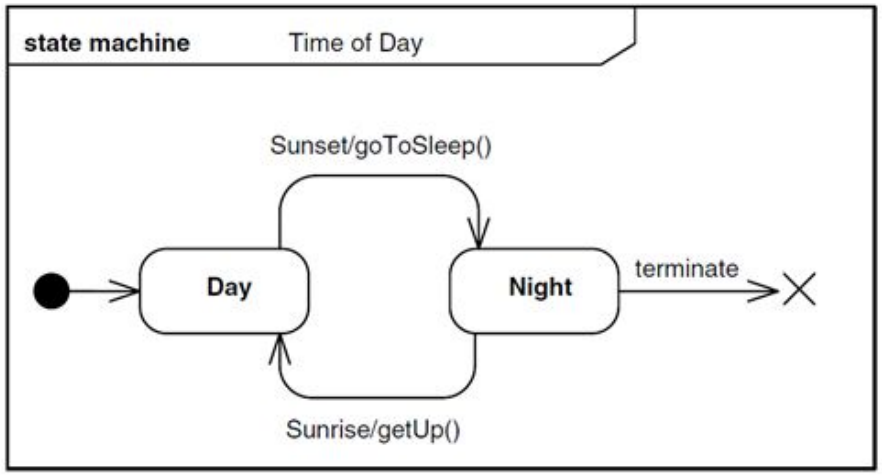
Forme particulière d'un état signifiant que la région ou l'état englobant sont considérés comme complet (la machine à état a atteint son état final)

Notation: 

Pseudo-Etat Terminate

Pseudo-Etat impliquant la terminaison de la machine à état (détruit l'objet)

Notation: 



Activités internes d'un Etat

- entry

  Comportement exécuté à l'entrée de l'état, avant toute activité interne

- do

  Comportement exécution déclenchée par une action ou un événement (après l'activité d'entrée et avant l'activité de sortie)

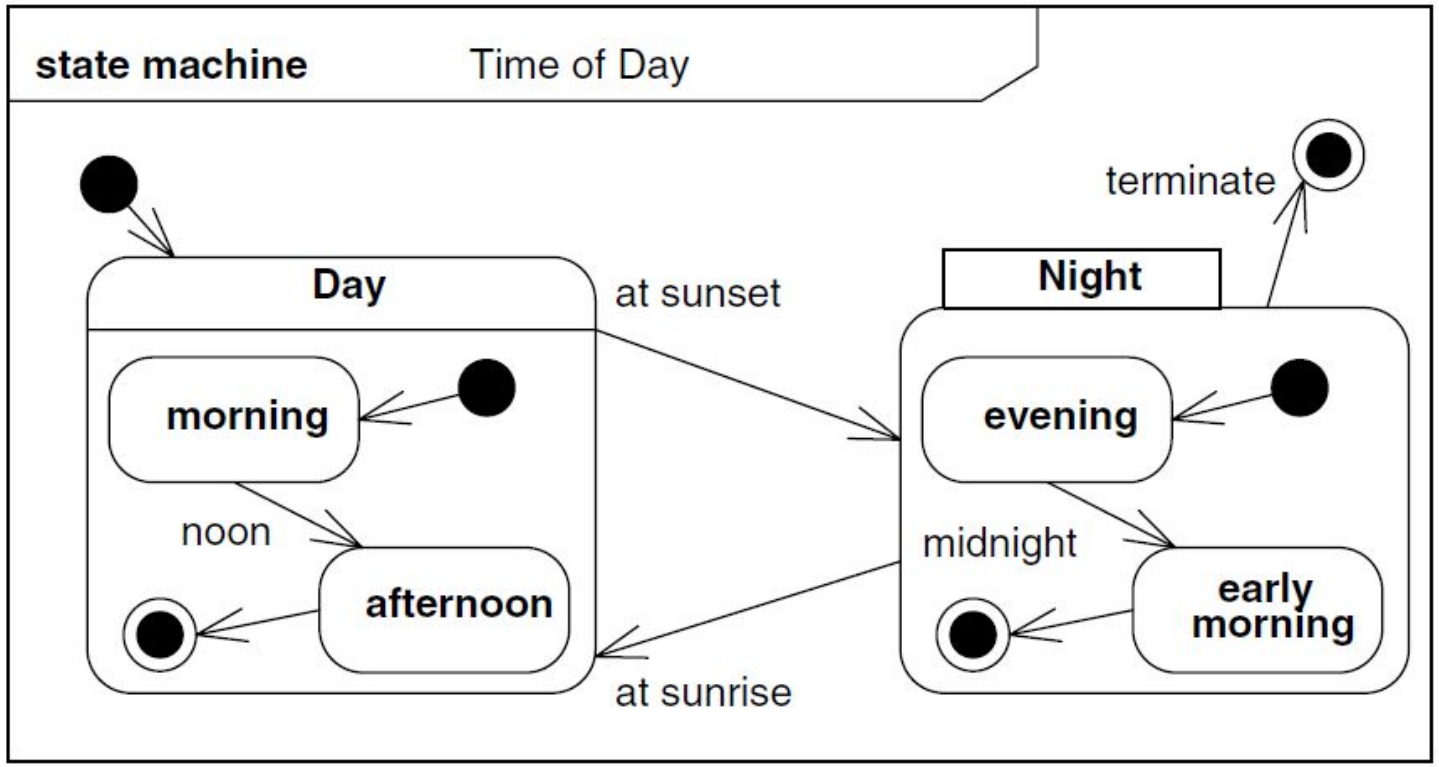
- exit

  Comportement exécuté quand l'objet quitte l'état, après le traitement de toute activité interne et avant toute activité de transition vers l'état suivant

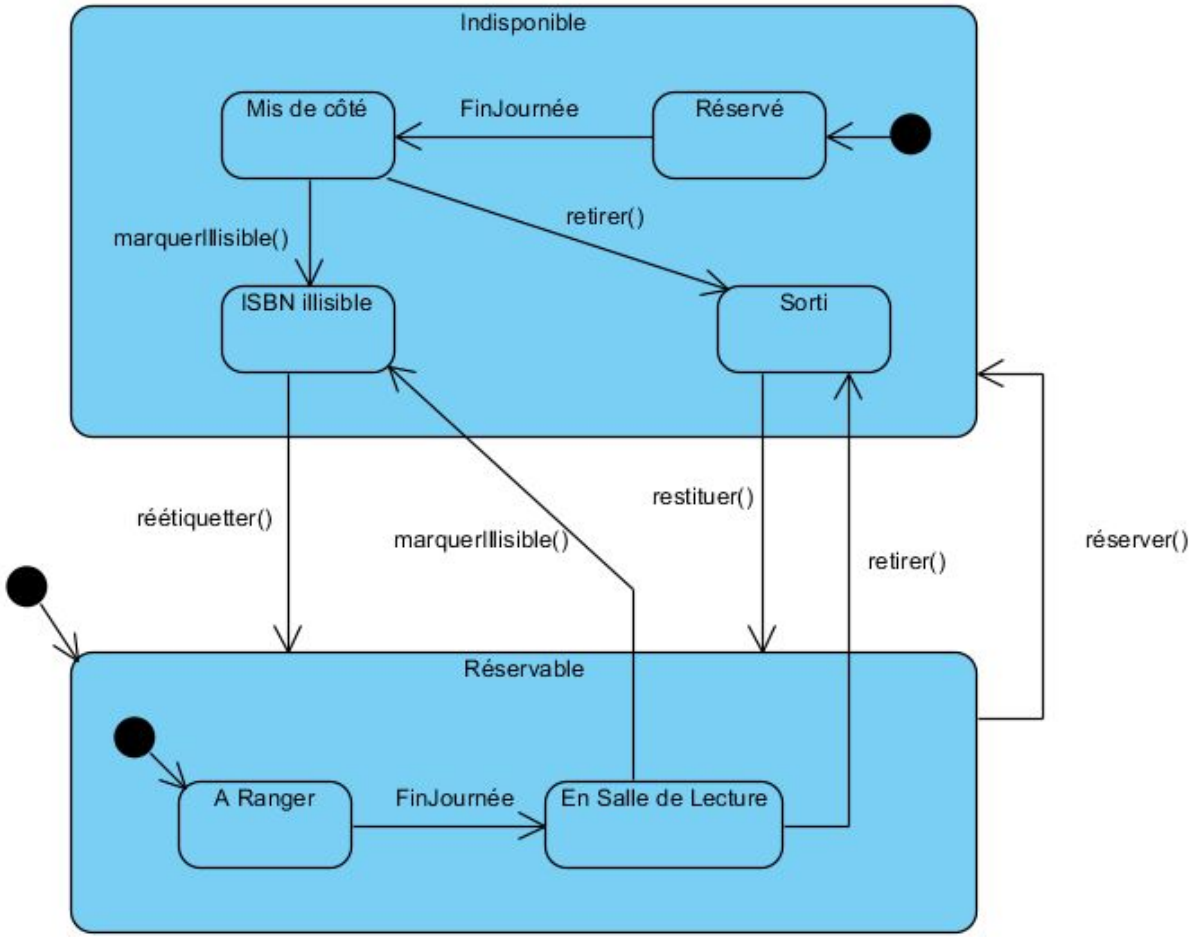
Les 3 types d'états

- Simple State
  - Etat sans sous-état
- Composite State
  - Simple composite State
    - Définition de sous-états et leurs transitions
  - Orthogonal State
    - Etat comprenant différentes régions indépendantes
- Submachine State

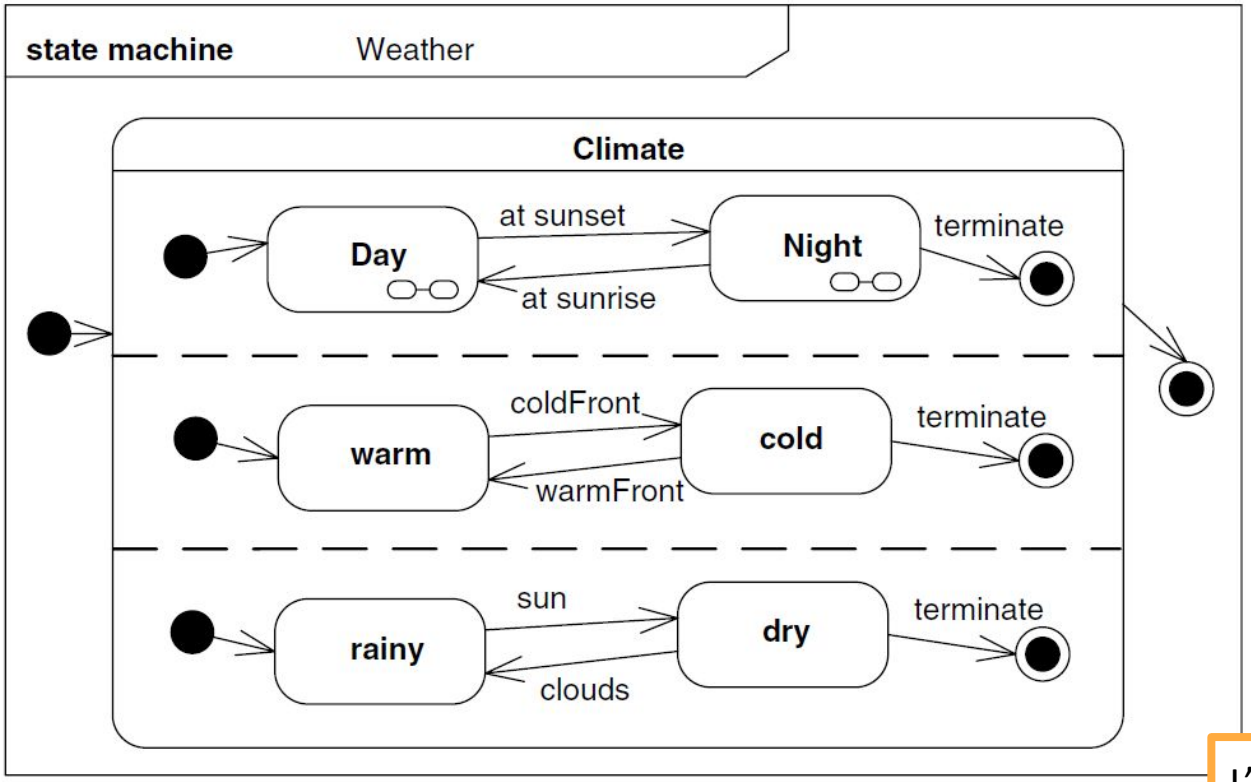
Composite State



Composite State



Etat Orthogonal



L' état orthogonal Climate est terminé lorsque tous les sous-états orthogonaux sont terminés

Exit Point

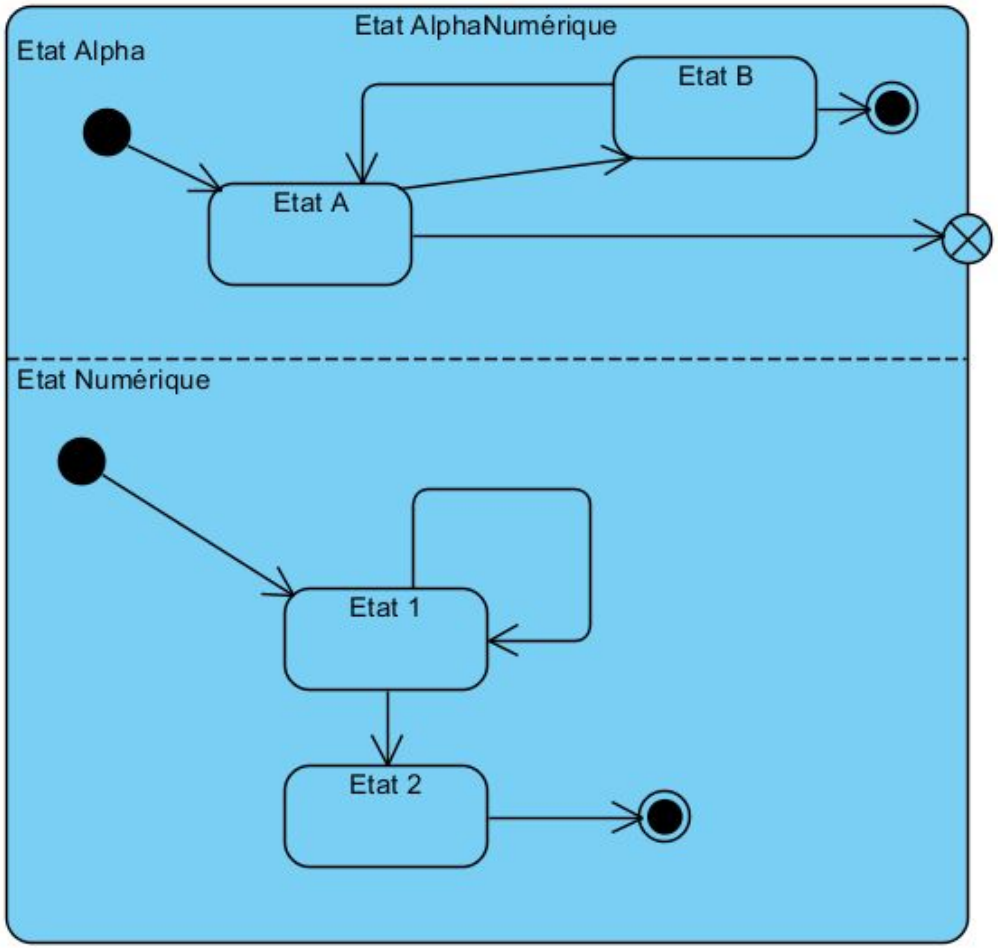
Les *Exit Points* sont uniquement utilisés dans les *Composite State* ou les *Submachine States*

Sémantique:

> Lorsque l'*Exit Point* d'une région est atteint, l'état composite est considéré dans son entièreté comme terminé (même si d'autres états orthogonaux ne sont pas terminés)

Utile pour les états orthogonaux

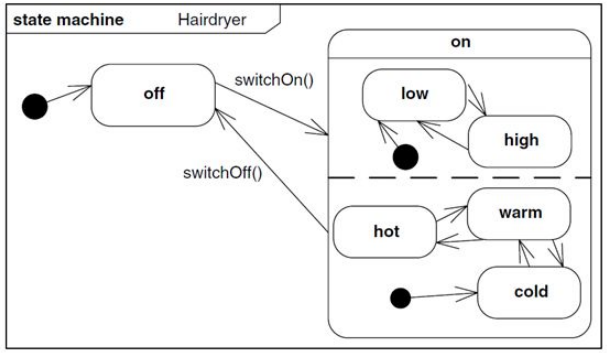
Exit Point

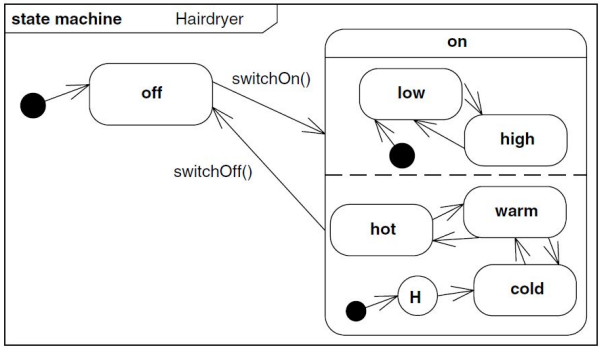

Etat AlphaNumérique
Etat Alpha
Etat B
Etat A
Etat Numérique
Etat 1
Etat 2

Shallow History

On entre dans le sous-état qui était actif lors de la dernière sortie de l'état étant 'Shallow History'

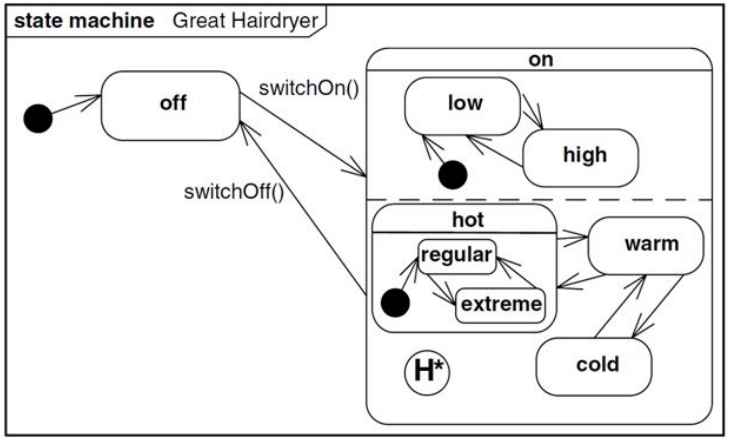
Le 'H' peut ne pas être "connecté" et se trouver simplement dans l'état ayant la caractéristique 'Shallow History'





Deep History

Identique au ‘Shallow History’, sauf que le mécanisme de mémorisation du dernier état est également valable pour les sous-états composites de l’état (=>tous les niveaux de profondeur)



Junction Point

Utiliser pour séparer ou rejoindre plusieurs transitions

Les éventuelles évaluations en cas de séparation d'une transition ne dépend pas de l'état source (>< point de décision)