IUT Nantes
Pôle Sciences et technologie
Nantes Université

R3.03 – Analyse

3_PHASE D’ANALYSE

Dalila TAMZALIT

IUT de Nantes – Département Informatique

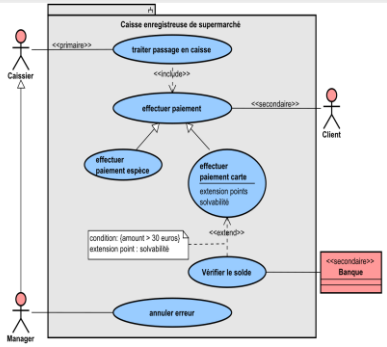
Analyse

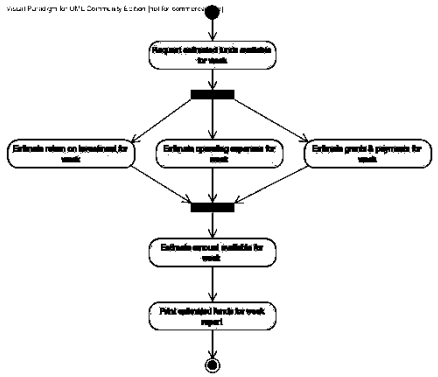
- Produire un **modèle d'analyse** du système qui soit
  - Correct, complet, cohérent et vérifiable
- But
  - Obtenir une compréhension méticuleuse des exigences
  - Décrire les exigences pour produire une conception et une implémentation **maintenables**
- Diffère de la cueillette des besoins
  - Structurer et formaliser les exigences
  - (Pas nécessairement compréhensible par le client)
- Force le client et les développeurs à prendre les **décisions** difficiles le plus tôt possible

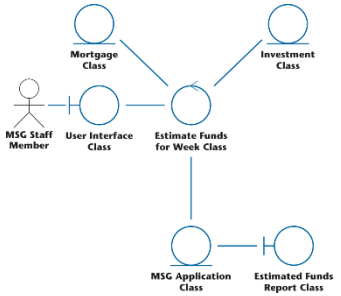
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Modèles d'analyse

- **Modèle fonctionnel**
  = Fonctionnalités du système
  → Cas d'utilisations, scénarios
- **Modèle dynamique**
  = Comportement du système
  → Activités, flux de données
- **Modèle d'objet**
  = Concepts individuels manipulés par le système et leurs propriétés
  → Classes, composants



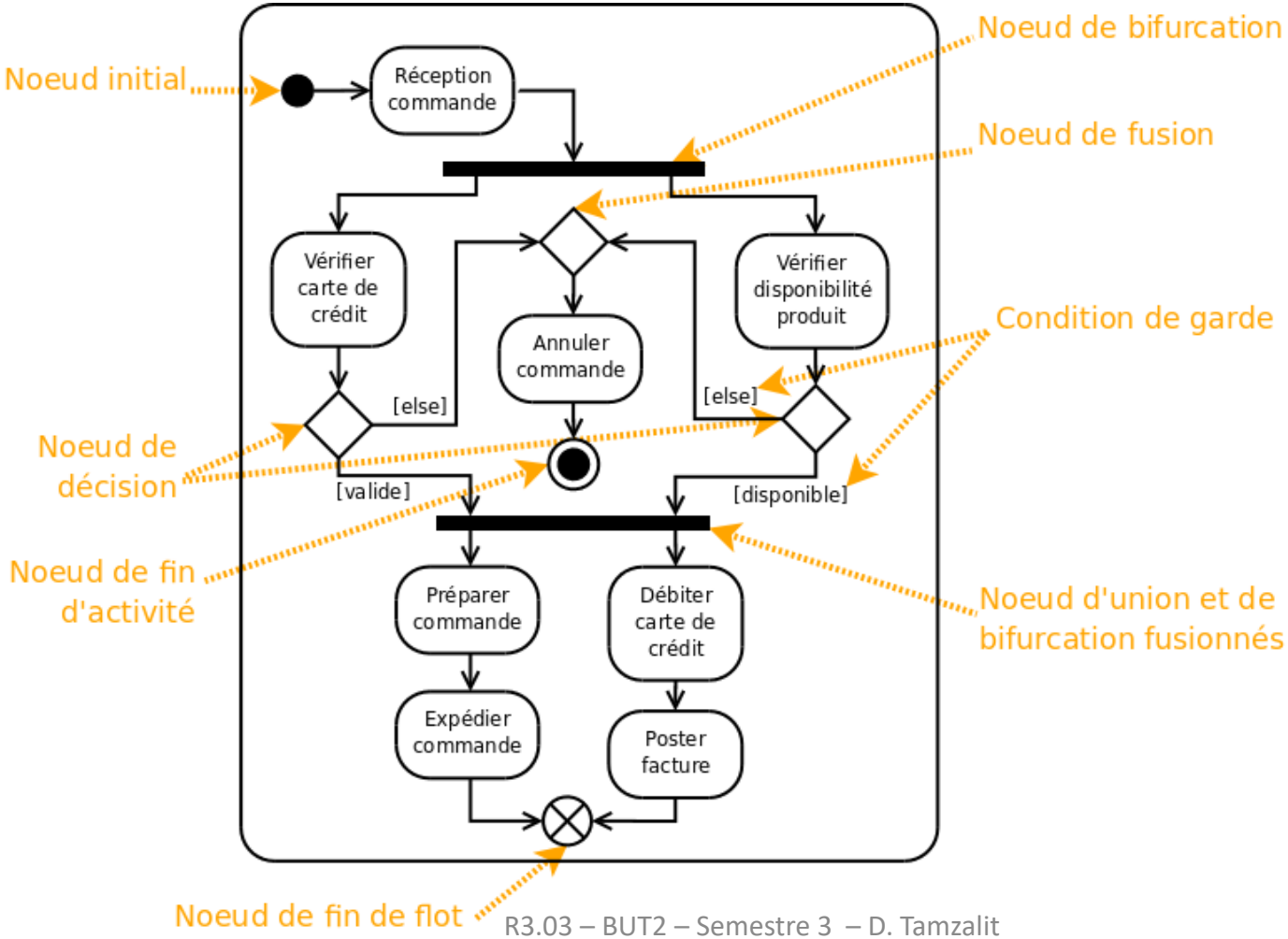




Modèle dynamique : modélisation du processus

- Identifier les activités nécessaires pour utiliser le logiciel
  - Définir les **étapes** du processus
  - Coordonner les différents **événements**
  - Identifier les responsabilités par **rôle**
- Pour cela on utilise un **diagramme d'activité UML**
  - Activités, actions
  - Transitions
  - Objets
  - Nœuds de contrôle
  - Partitions

Diagramme d'activité UML



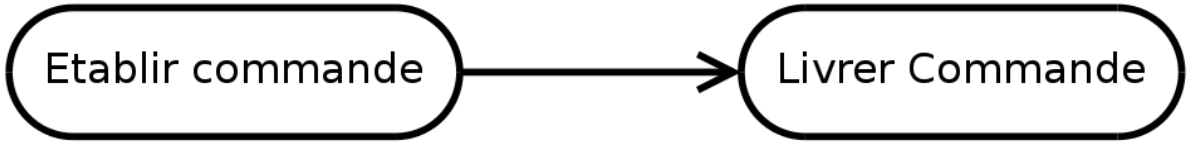
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Action

- Plus petit **traitement** en UML qui a une incidence sur l'état du système
  - Affectation d'attributs
  - Accès à une propriété structurelle
  - Création d'objet
  - Calcul simple
  - Appel d'opération ou d'événement
  - Acceptation d'opération ou d'événement
  - Levée d'exception
- Activité est un comportement **complexe**
  - Terme **abstrait** représentant un séquencement d'actions

Transition

- Passage d'une activité à une autre
  - Déclenchée quand l'activité source est terminée
  - Provoque le début de la prochaine activité cible
- Contrairement aux activités, les transitions sont **atomiques**
  - Les activités ont une durée donc peuvent être interrompues (pas les transitions !)



Événements

- Événements **externes**
  - Survient à l'extérieur du système (par un acteur)
    - Client passe une commande
- Evénements **temporels**
  - Attente dans le temps
    - À chaque semaine
- Événements **d'état**
  - Survient à l'interne et déclenche un besoin de traitement
    - Rupture de stock

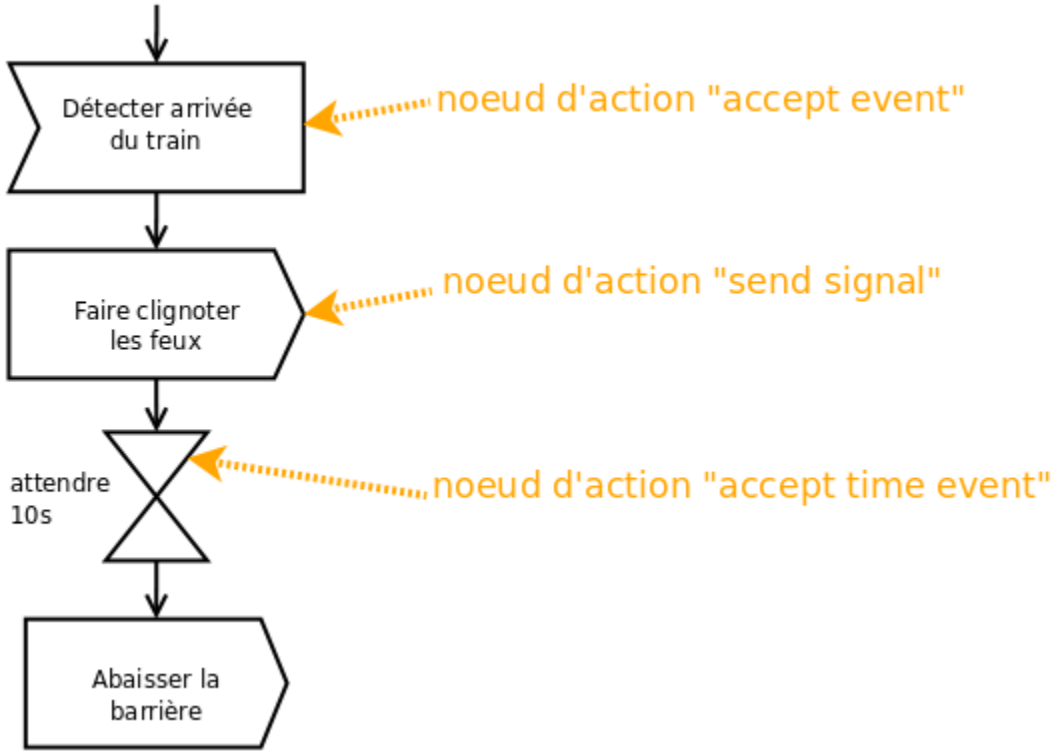
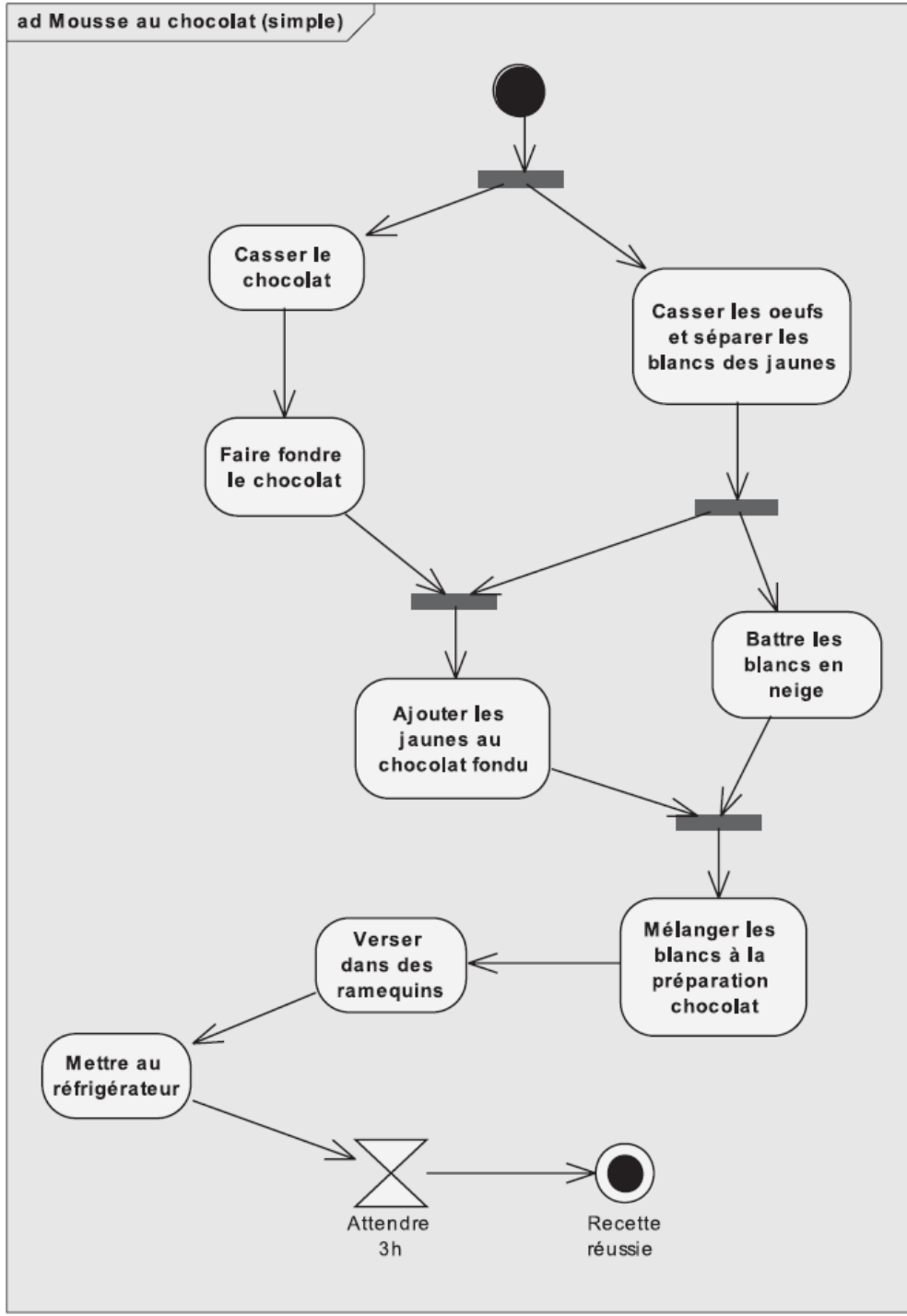
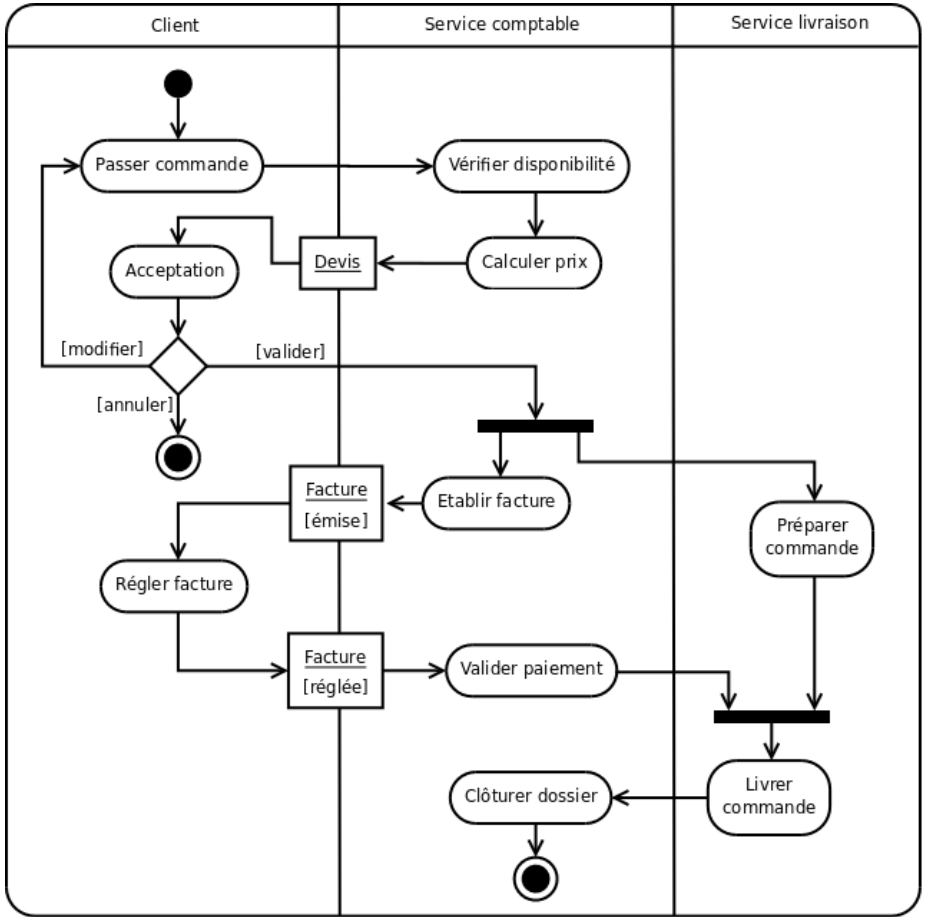


Diagramme d'activités

- Comment faire une mousse au chocolat ?
  - Recette simplifiée : commencer par casser le chocolat en morceaux, puis le faire fondre.
  - En parallèle, casser les œufs en séparant les blancs des jaunes.
  - Quand le chocolat est fondu, ajouter les jaunes d'oeuf.
  - Battre les blancs en neige jusqu'à ce qu'ils soient bien fermes.
  - Les incorporer délicatement à la préparation chocolat sans les briser.
  - Verser dans des ramequins individuels.
  - Mettre au frais au moins 3 heures au réfrigérateur avant de servir



Avec partitions et rôles



Utilisation des diagrammes d'activité

- Met l'accent sur les traitements
  - Flots de **contrôle** et de **données**

- Illustre et consolide description textuelle des CU
  - Modélisation du **workflow** de chaque scenario
  - Concentre sur les activités **vues par les acteurs**

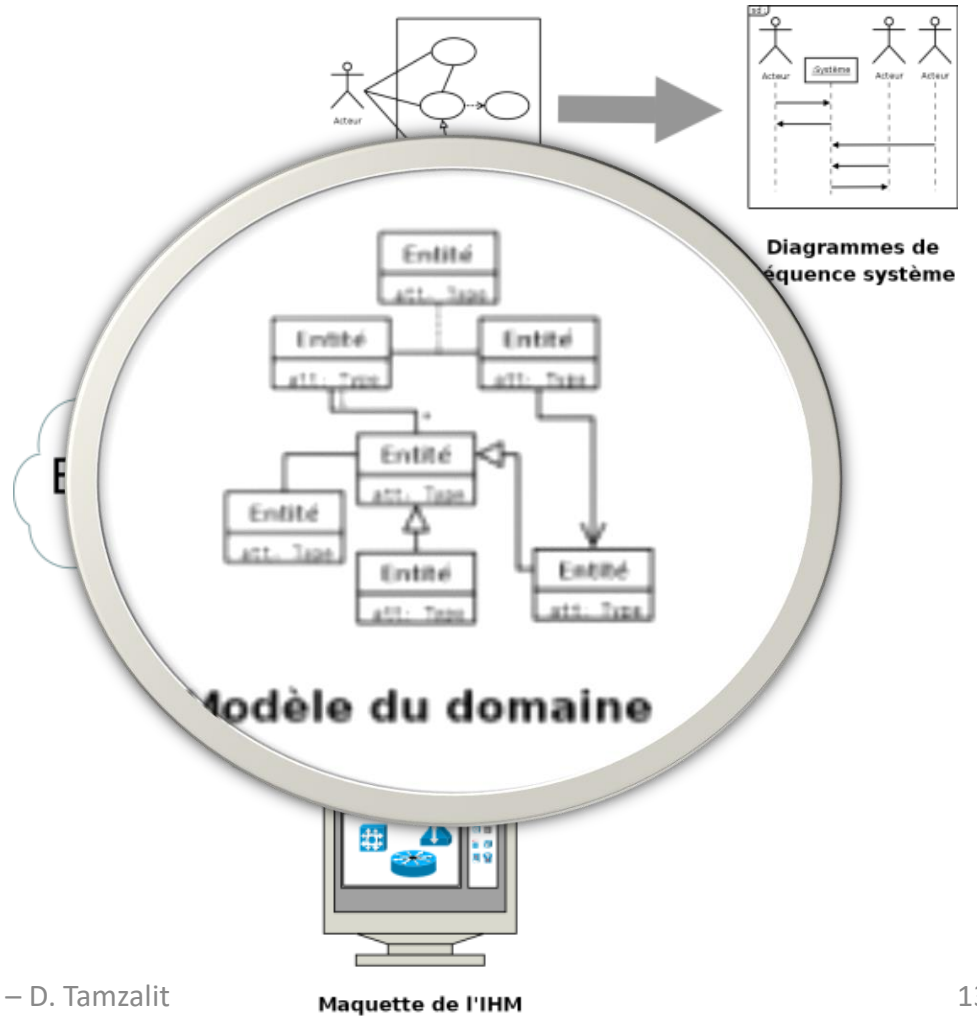
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Modèle des classes du domaine

Identifier les objets (et classes) est la tâche la plus difficile de la **conception orientée objet**

Modèles en phase d'analyse

- La modélisation des besoins par des cas d'utilisation revient à une analyse fonctionnelle classique.
- La phase d'analyse du domaine permet d'élaborer la première version du diagramme de classes.
- L'élaboration du modèle des classes du domaine permet d'opérer une transition vers une véritable modélisation objet.

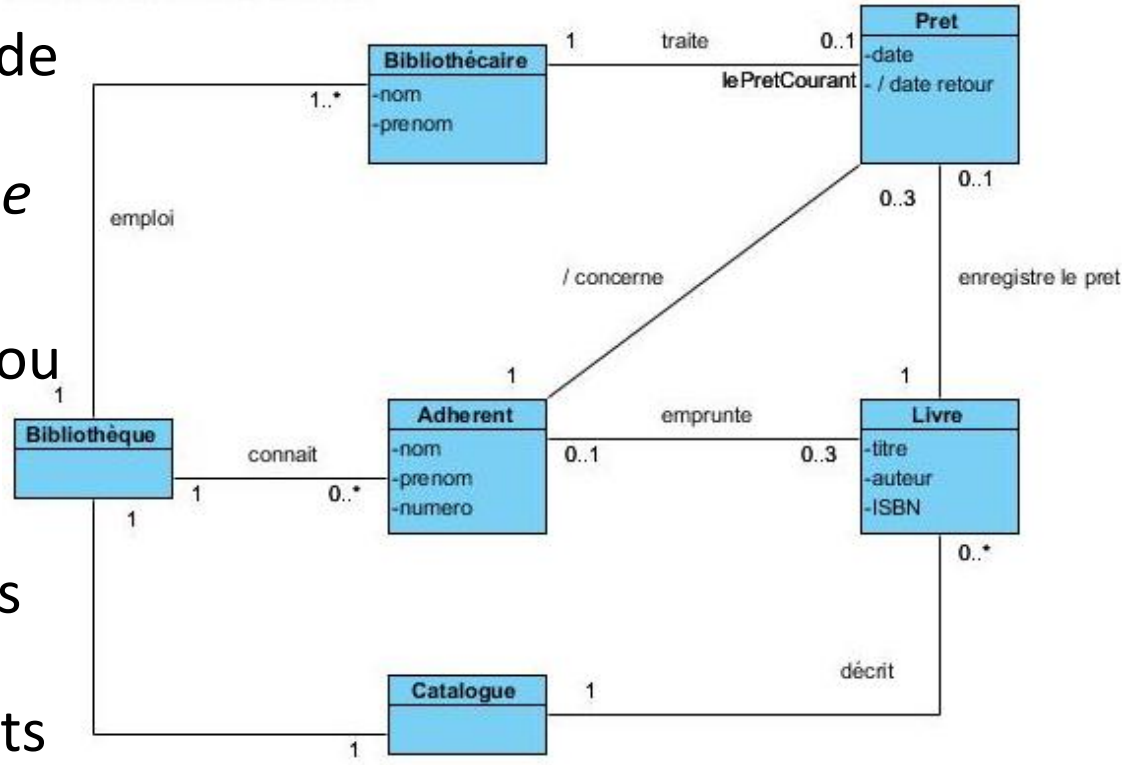


Approches pour l'identification

- Analyse basée sur les **scénarios**
  - Identifier les objets, leurs attributs et méthodes par scénario
- Approche **grammaticale**
  - Substantifs et verbes
- Baser l'identification sur les choses **tangibles** du domaine
  - Analyse du domaine
  - Structures de données qui leur sont appropriées
- Approche **comportementale**
  - Identifier les objets selon ce qui participe à chaque comportement du système

Modèle des classes du domaine

- La phase d'analyse du domaine permet d'élaborer la première version du diagramme de classes appelée *modèle du domaine* ou *diagramme de classes d'analyse* ou *diagramme de classes de conception générale*.
- Il définit les classes qui modélisent les entités ou concepts présents dans le domaine (on utilise aussi le terme de métier) de l'application.
- Ces entités ou concepts peuvent être identifiés directement à partir de la connaissance du domaine ou par des entretiens avec des experts du domaine (confère l'ingénierie des exigences).

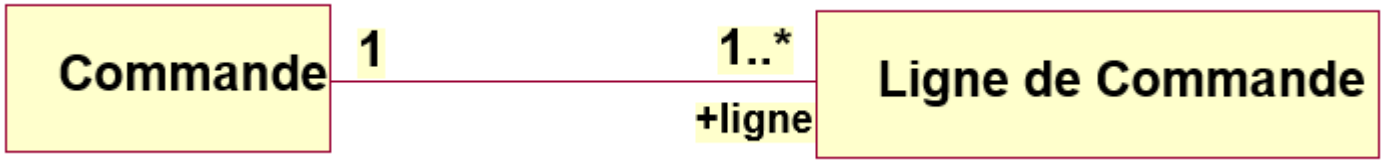


Modèle des classes du domaine

- Il faut absolument utiliser le vocabulaire du métier pour nommer les classes et leurs attributs.
- Les classes du modèle du domaine ne doivent pas contenir d'opération, mais seulement des attributs.
- Les étapes à suivre pour établir ce diagramme sont :
  - identifier les entités ou concepts du domaine ;
  - identifier et ajouter les associations et les attributs ;
  - organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant l'héritage ;
  - le cas échéant, structurer les classes en paquetage selon les principes de cohérence et d'indépendance.

Concept de rôle

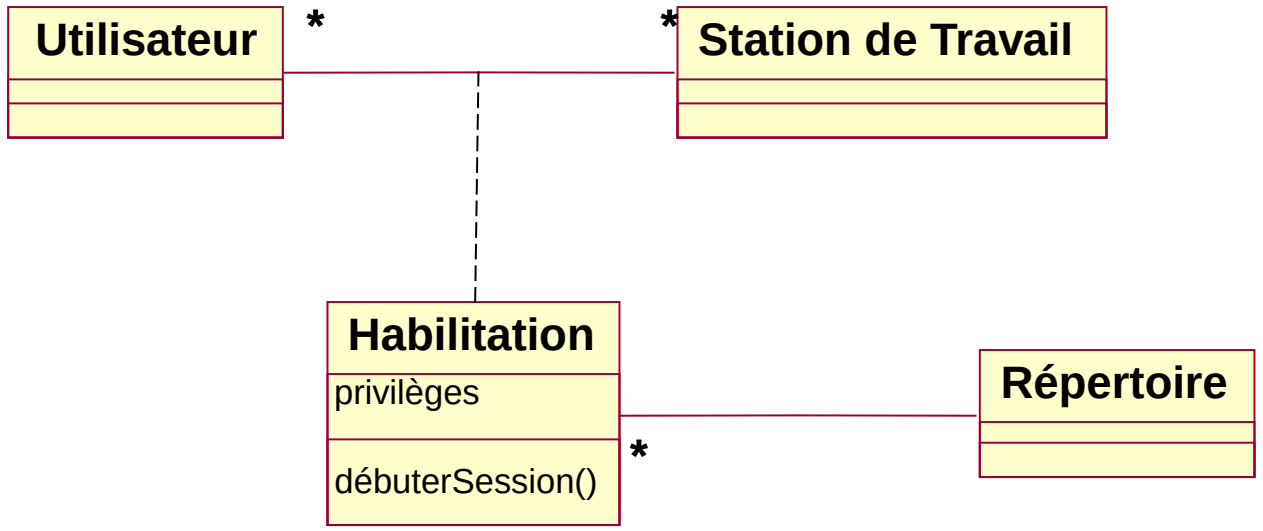
- Rôles : rôle que joue une classe dans une association.
- Aux deux extrémités d'une association.
- Nom de rôle : le rôle est nommé explicitement.
- Exemple : ligne de l'extrémité Ligne de Commande associée à Commande.



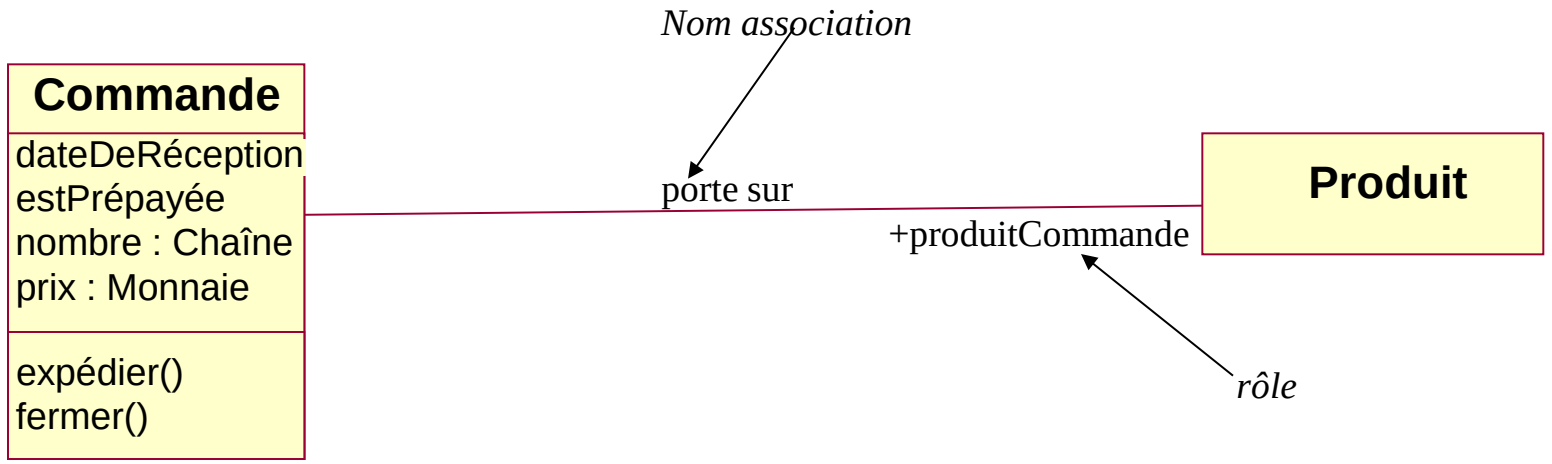
- En l'absence d'étiquette, le nom de rôle est le nom de la classe cible.

Classe-association

- Une association peut être représentée par une classe si elle est porteuse d'attributs/opérations.
- C'est une classe comme les autres. Elle peut avoir des relations.

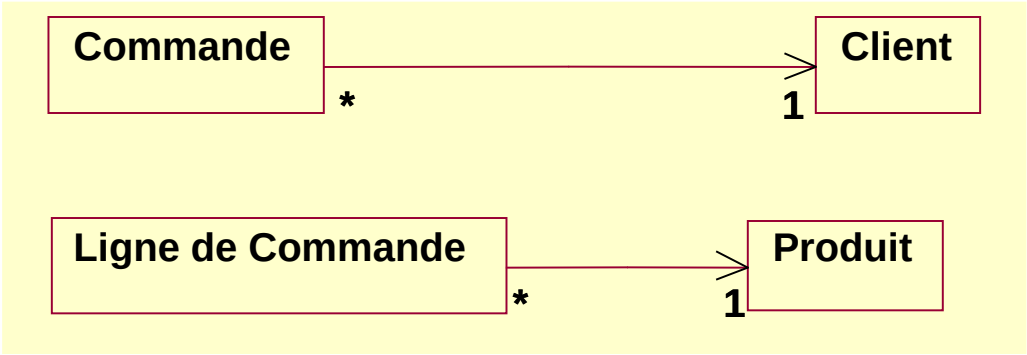


Associations et nom de rôle



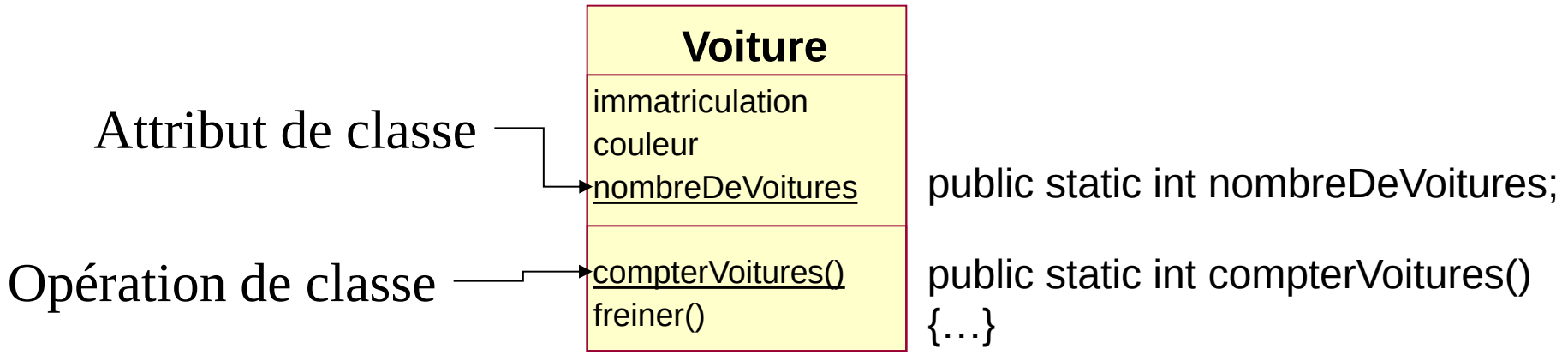
Associations

- Navigabilité :



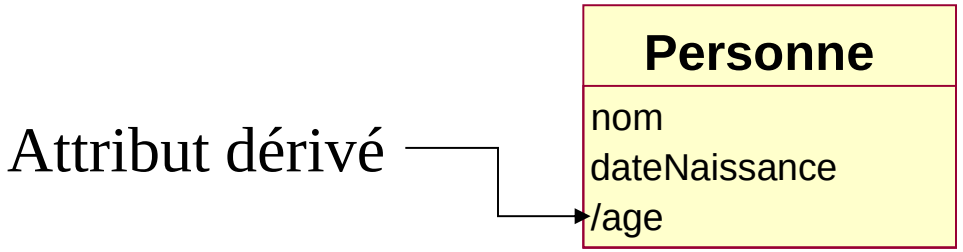
  - Association unidirectionnelle.
    - Une Commande peut informer de ses Clients ; mais le client ne peut pas indiquer ses commandes.
  - Importance pour l'implémentation et non pour l'aspect purement conceptuel.
    - Commande aura un pointeur sur Client mais Client n'aura pas de pointeurs sur Commande.
- Association sans navigabilité :
  - Bidirectionnelle ou Inconnue.
  - Les navigations sont l'inverse l'une de l'autre.

Attributs/opérations de classe



Attributs dérivés

- Attribut dérivé: attribut qui se calcule



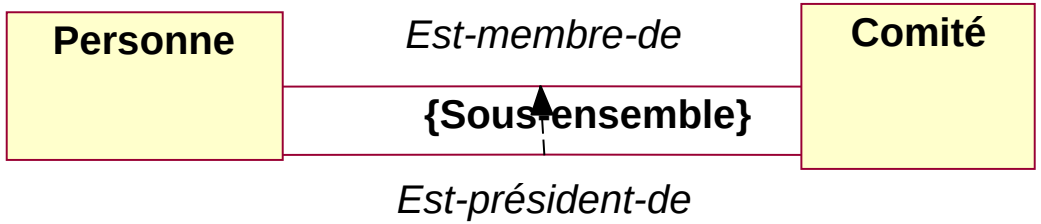
IUT Nantes
Pôle Sciences et technologie
Nantes Université

Contraintes

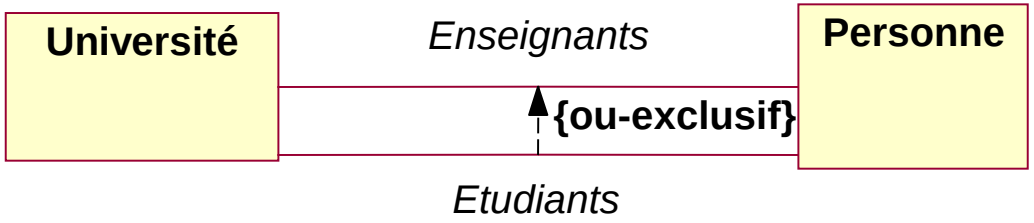
- Associations, attributs et généralisation :
  - Expriment les contraintes importantes
  - Ne peuvent pas toutes les indiquer. Il faut cependant les capturer. UML fait le choix de les exprimer au niveau du diagramme de classes.
- Contraintes :
  - Exprimées entre accolades.
  - En langage naturel ou en OCL (Object Constraint Language).
  - Généralement implémentées sous forme d'assertions dans le langage de programmation.

Quelques contraintes

- Contrainte de sous-ensemble : {sous-ensemble}
  - Une collection est incluse dans une autre collection.



- Contrainte du ou-exclusif : {ou-exclusif}
  - Pour un objet, une seule association valide parmi un groupe d'associations.



  - Évite l'introduction de sous-classes artificielles pour représenter l'exclusivité.

Généralisation

- Généralisation :
  - Relation non-symétrique.
  - Relation entre une classe et une ou plusieurs de ses versions plus raffinées ou plus spécialisées.
  - Relation de classification entre un élément plus général et un élément plus spécifique.
- Héritage :
  - Mécanisme objet qui permet aux classes de partager des attributs et des opérations en se basant sur une relation, habituellement la généralisation.

Super-classe et Sous-classe

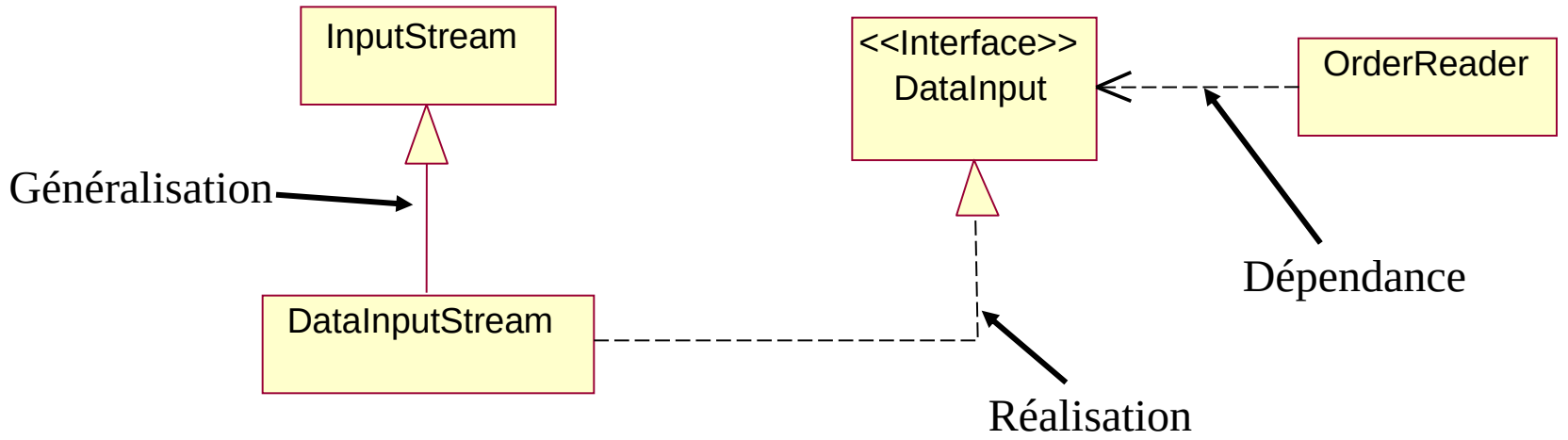
- Super-classe :
  - Contient les attributs et les opérations communs.
- Sous-classe :
  - Raffine ou restreint les attributs et opérations hérités.
  - Ajoute ses propres attributs et opérations.
- Une sous-classe hérite de sa super-classe :
  - Attributs
  - Opérations
  - Associations
- Héritage multiple :
  - Une classe hérite de ses super-classes.
  - Ses attributs et opérations sont ceux de ses parents.
  - UML ne fournit pas de règles de résolution de conflits.

Interfaces /Classes abstraites

- Classe abstraite : nom en italique ou la contrainte {abstract}
- Interface :
  - Utilise un type pour décrire le comportement visible d'une classe, sans implémentation.
  - Est un stéréotype.
- Une classe peut :
  - Spécialiser une classe abstraite.
  - Réaliser une interface.

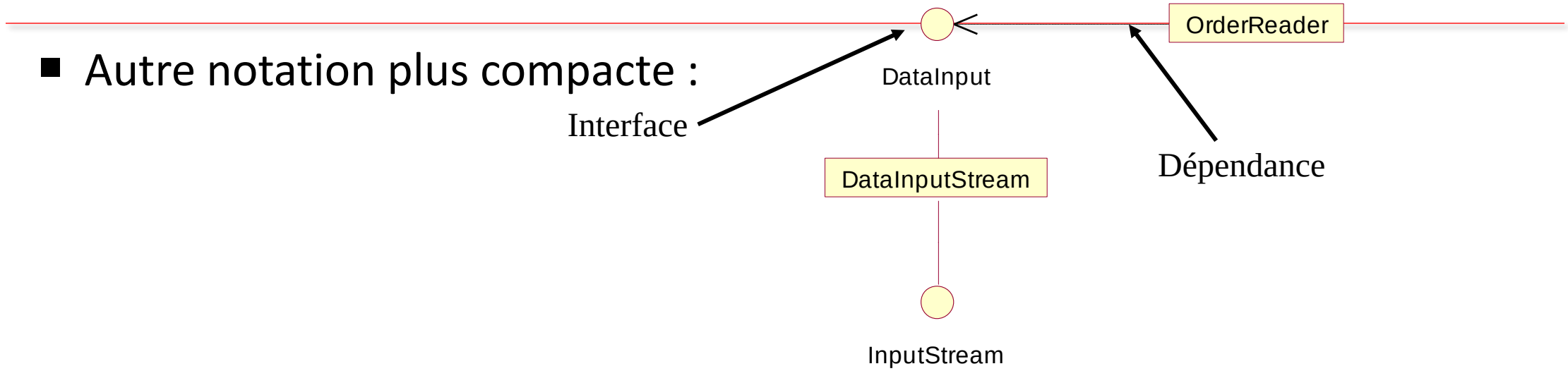
Interfaces - Classes abstraites

- En UML, le symbole de la réalisation est délibérément semblable à la généralisation.
- Réalisation : une classe implémente le comportement spécifié par une autre. Une classe d'implémentation peut en réaliser une autre : elle se conforme à son interface sans en hériter forcément.



Interfaces - Classes abstraites

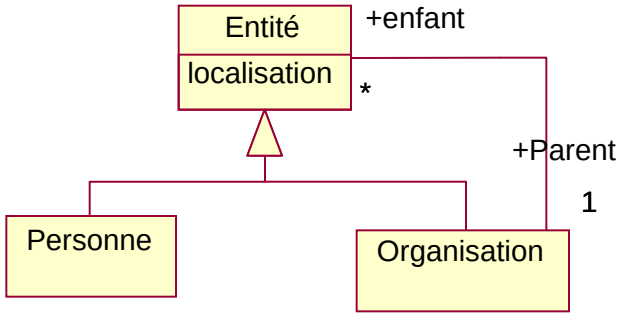
- Autre notation plus compacte :



- la réalisation des interfaces est représentée par de petits cercles, rattachés à la classe qui réalise l'interface.
- Ne distingue pas entre réalisation d'interface et sous-classement de classe abstraite.
- Mais ne permet pas de représenter les opérations des interfaces ni les relations de généralisation entre elles.

Diagramme d'objets

- Un diagramme d'objets ou d'instances est un instantané des objets présents à un moment donné.
- Diagramme de collaborations dépourvu de messages.
- Utilité : pour représenter une configuration particulière d'objets, surtout quand les liens possibles entre objets sont complexes :



Structure d'une entité

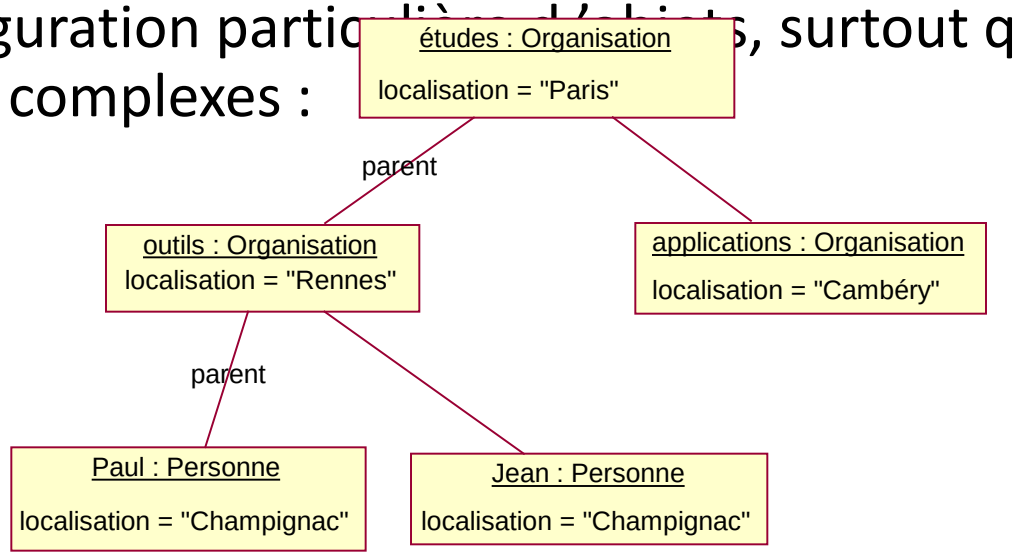
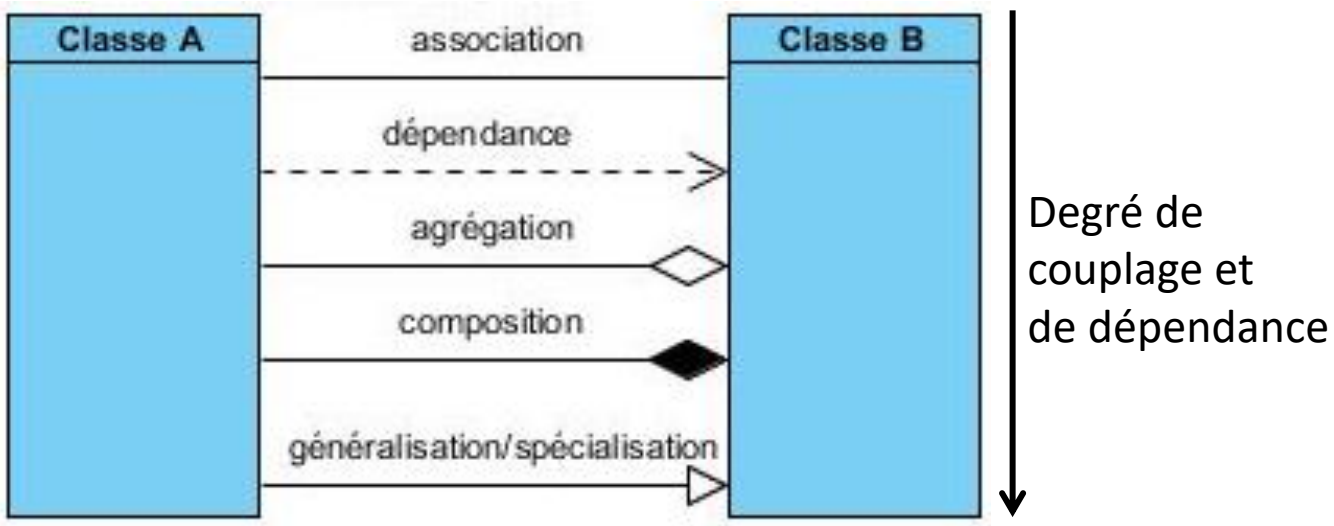


Diagramme d'objets : exemples d'instances d'une entité

Associations & Relations

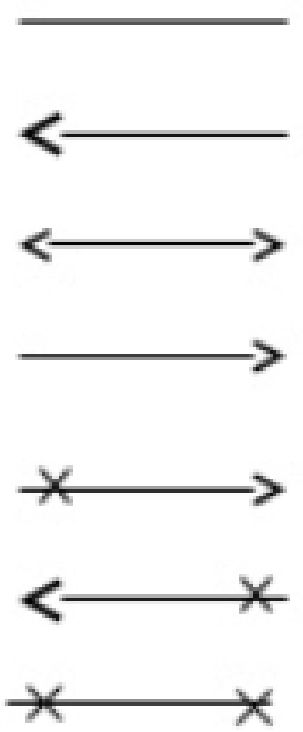
- Au cœur des phases d'analyse et de conception
- **Différents types de relations :**
  - **entre classes :**



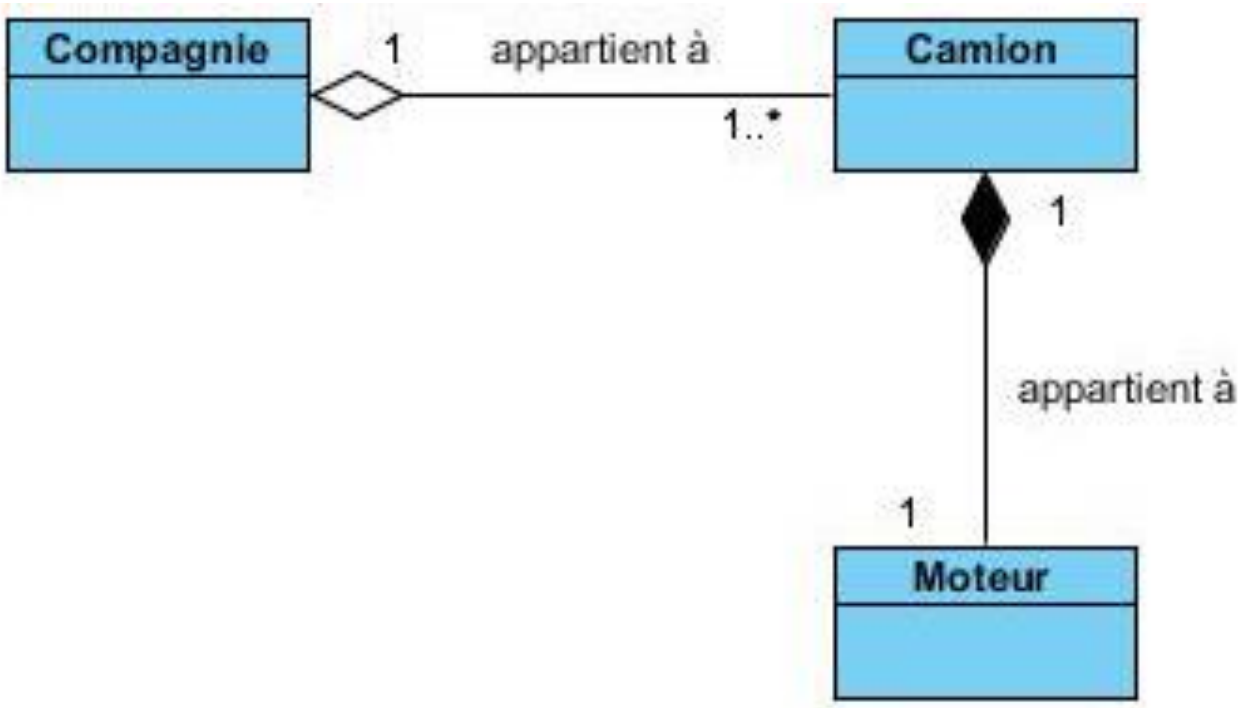
  - **entre classe et interface :** implémentation & association
  - **entre interfaces :** généralisation

Association et navigation

- En cas de doute, ne mettre aucune flèche

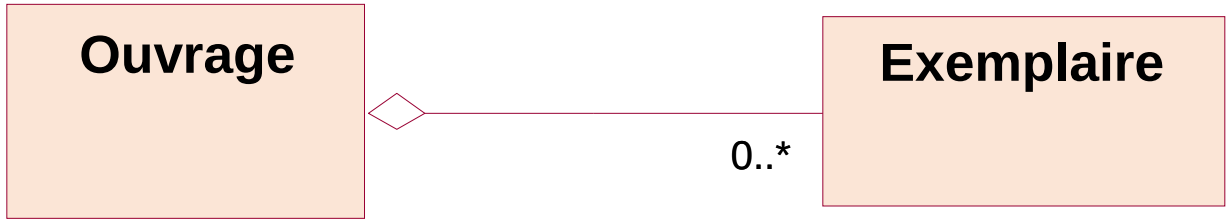


Agrégation et Composition



Agrégation

- Une association exprimant un couplage fort lié à une relation de subordination : fort couplage logique
- Bidirectionnelle et asymétrique.
- **Attention** :
  - Un élément agrégé peut être lié à d'autres classes
  - La suppression de l'ensemble n'entraîne pas celle de l'élément
- **Exemple** : un ouvrage (*agrégat*) et ses exemplaires (*composant*).

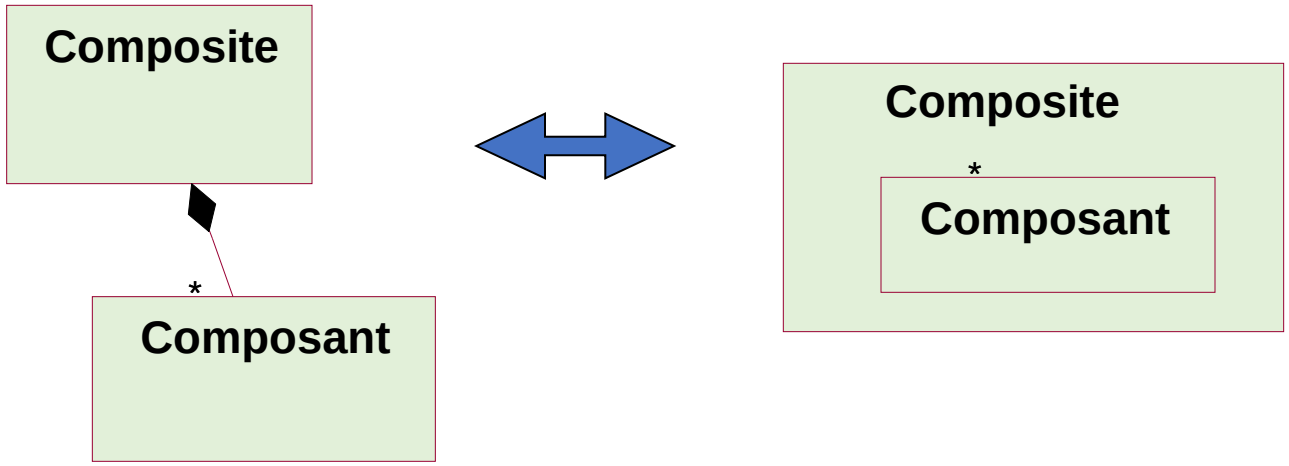


Composition

- La composition est
  - une agrégation forte
  - lie les cycles de vie entre le composé ou composite (ensemble) et les composants (éléments)
- Les règles supplémentaires obligatoires pour la composition sont les suivantes :
  - la suppression du composé entraîne la suppression des composants
  - les composants sont créés par le composé
  - Un composant est lié à un composé unique

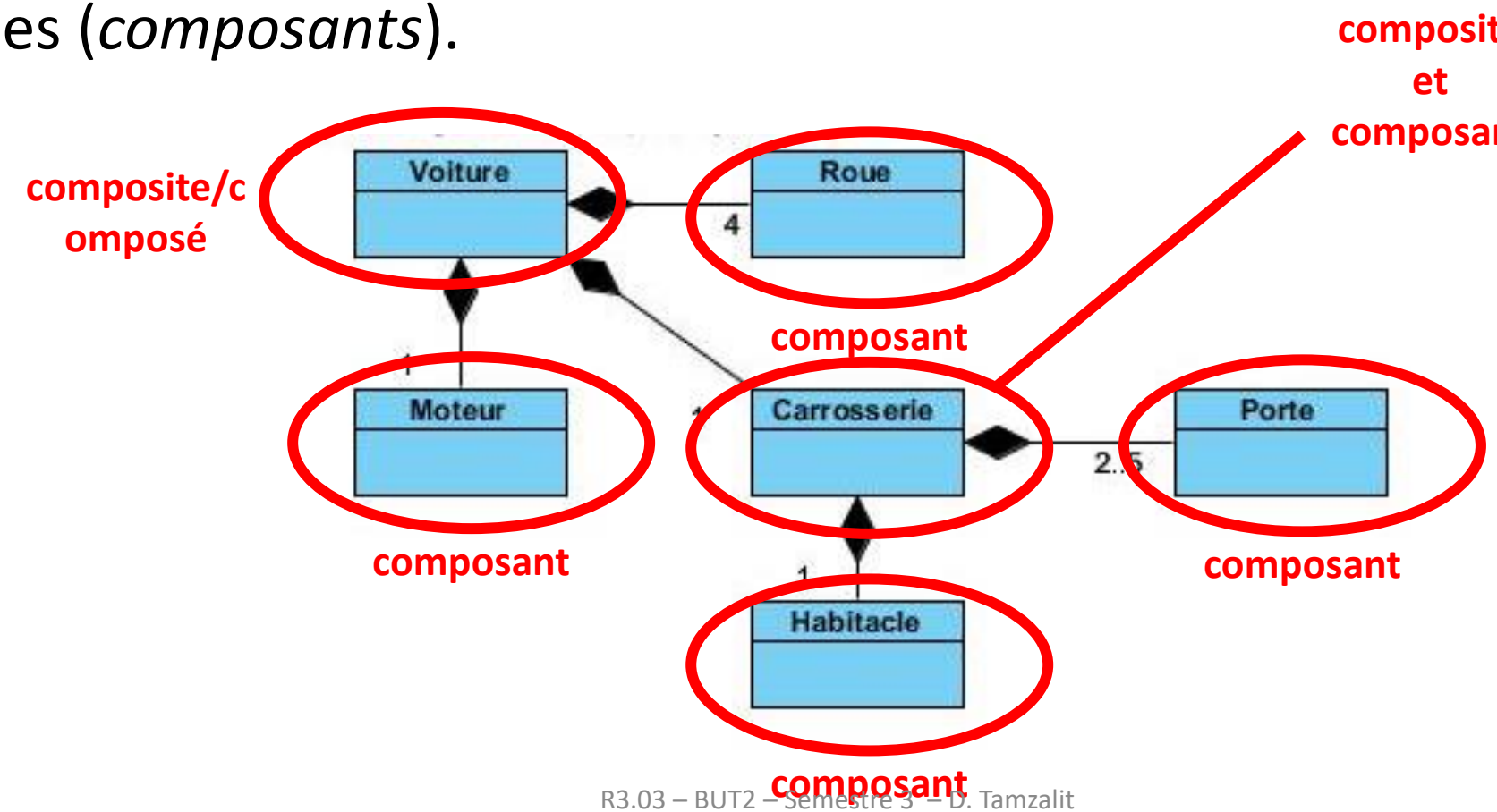
Composition

- Exprime souvent une composition physique.
- UML offre deux possibilités de représentation :

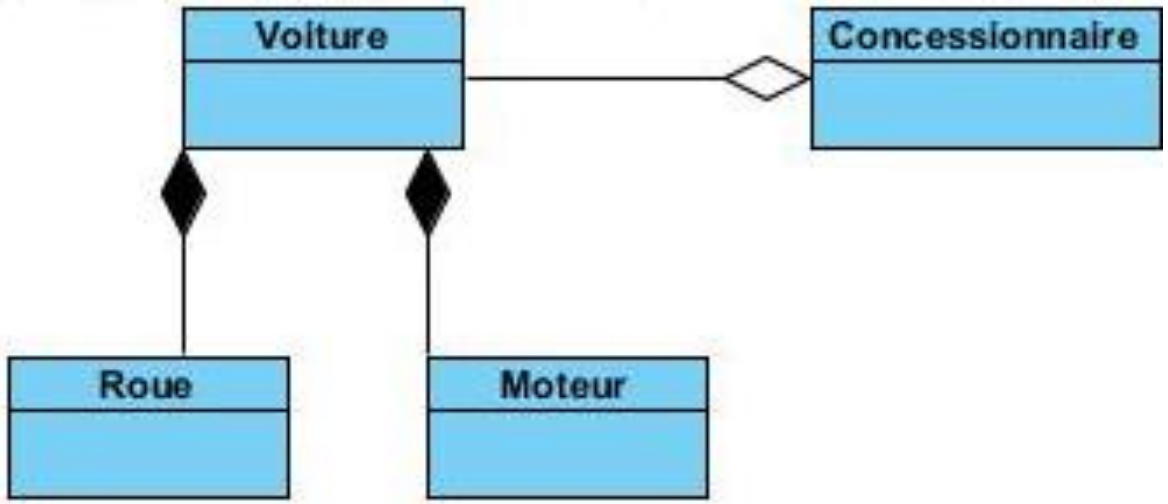


Composition

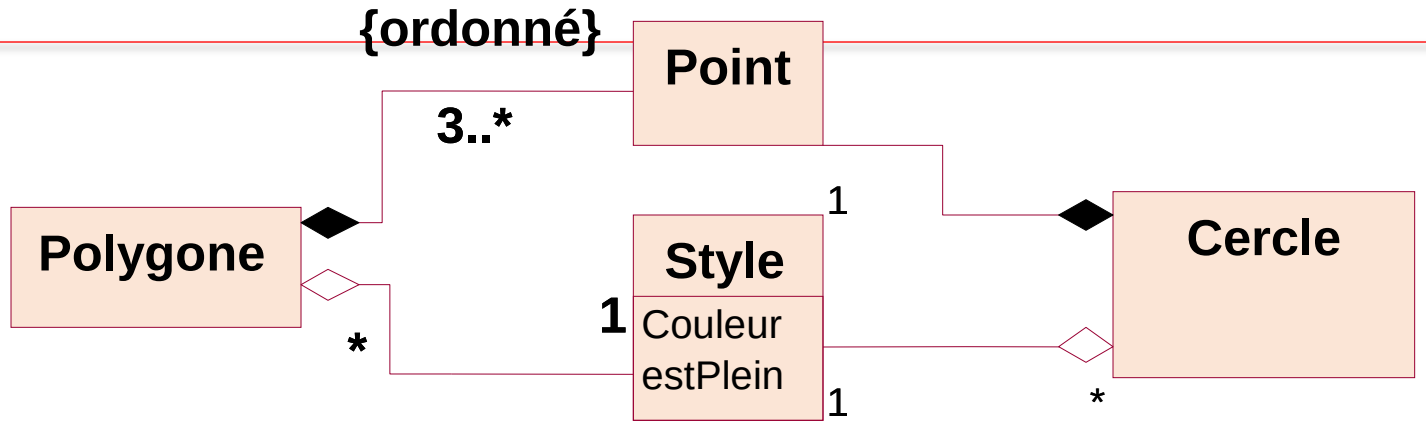
- Exemple : une voiture (*composite*) et son moteur, sa carrosserie et ses roues (*composants*).



Agrégation et composition



Agrégation et Composition



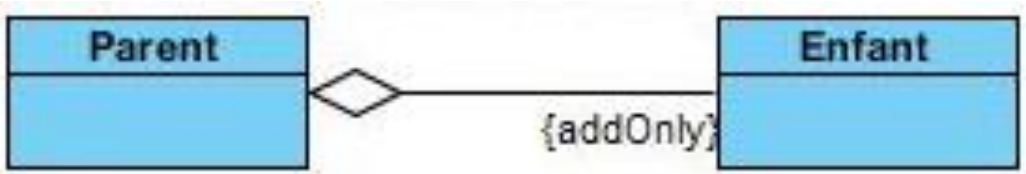
- Un Polygone ainsi qu'un Cercle sont composés de points :
  - La destruction du composite implique la destruction de la composition.
  - La destruction du composant implique celle du composite.
- Un Style peut concerner plusieurs Polygones et/ou Cercles :
  - Style est indépendant de l'existence d'agrégats.
  - Sa modification implique forcément celle de ses agrégats.

Agrégation & composition vers le Code

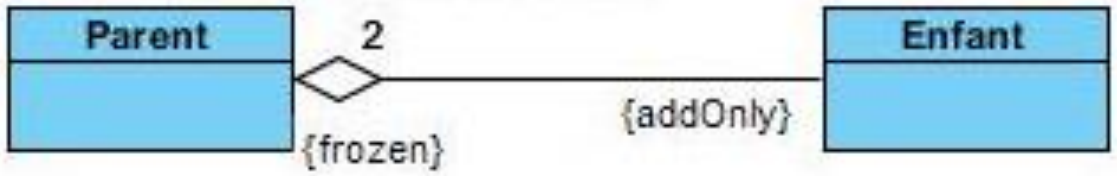
- Agrégation = Association
  - Règles de passage au code identiques,
- Composition = association + dépendance forte de la vie des composants vis-à-vis de leur composite
  - Règles de passage au code identique à une association
  - **+ prévoir un destructeur !**

D'autres contraintes entre associations

- *{addOnly}* : autorise l'ajout de nouveaux objets mais pas leur suppression ni leur mise à jour.



- *{Frozen}* : interdit l'ajout, la suppression ou la mise à jour des liens d'un objet vers les objets de la classe associée.



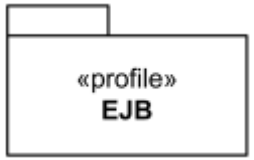
Stéréotype

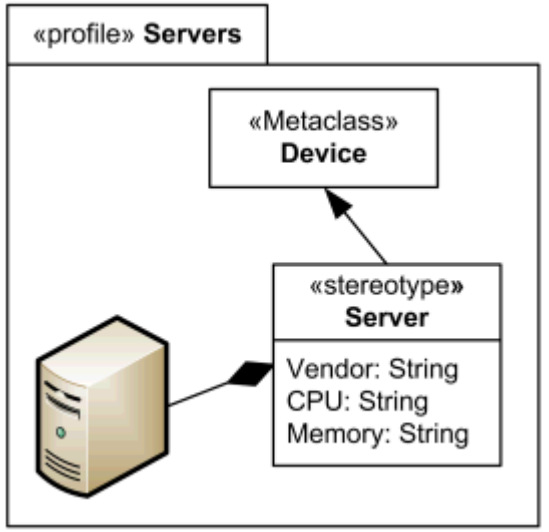
- Principal mécanisme d'extension d'UML.
- Permet de construire de nouveaux types, par exemple des objets affichables.
- Généralement représenté par un texte entre guillemets
  - Exemple `<<Objets Dessinables>>`

  et une icône associée.

- Stéréotypes de classe, d'association ou de généralisation.

Exemples stéréotypes





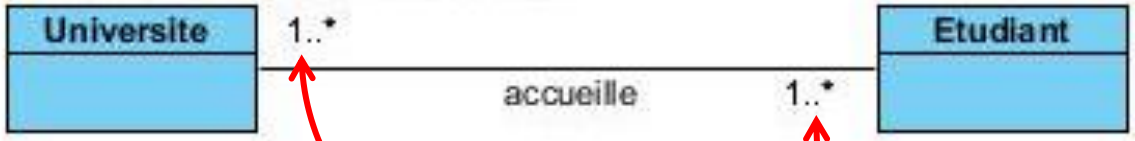
Qualification

- Une association peut être qualifiée.
- Restriction qui consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association
  - Restriction de la multiplicité

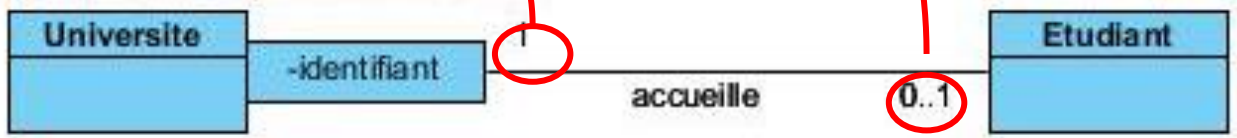
- La paire (instance de la classe A, valeur de la qualification) identifie un sous-ensemble des instances de la classe B.
- La clé est représentée sur le rôle de la classe de départ, dans un rectangle.

Exemples qualification

- Une université peut accueillir plusieurs étudiants et un étudiant peut être inscrit à plusieurs universités.



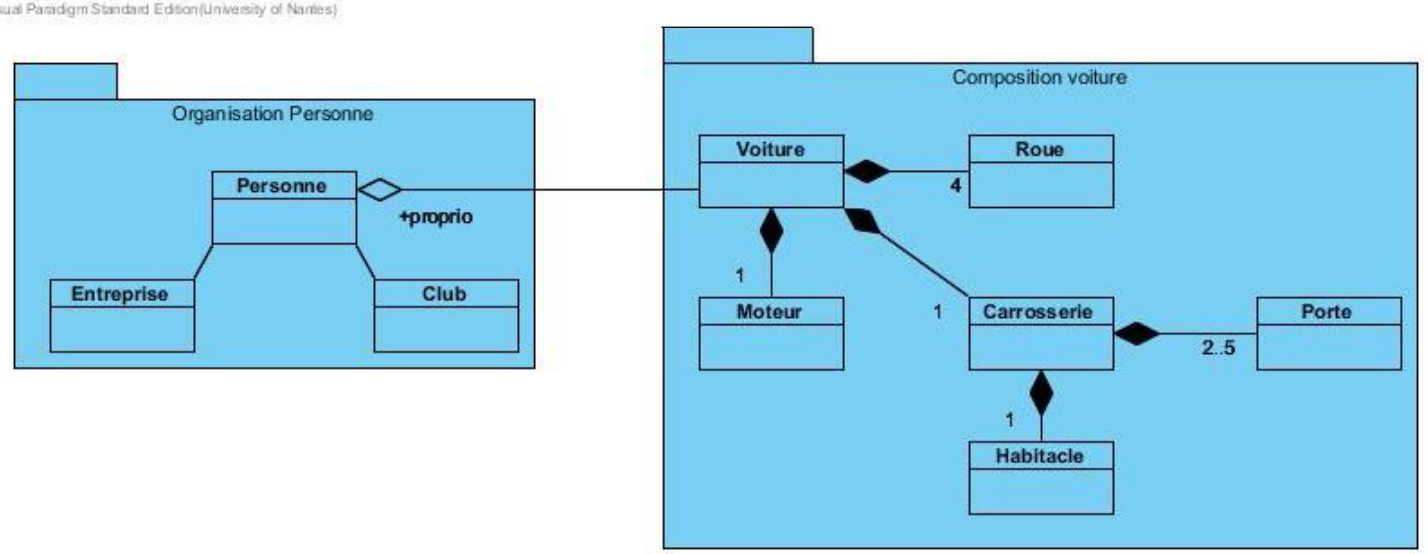
- Un étudiant a un identifiant unique dans une université qui l'accueille



- Clés multiples

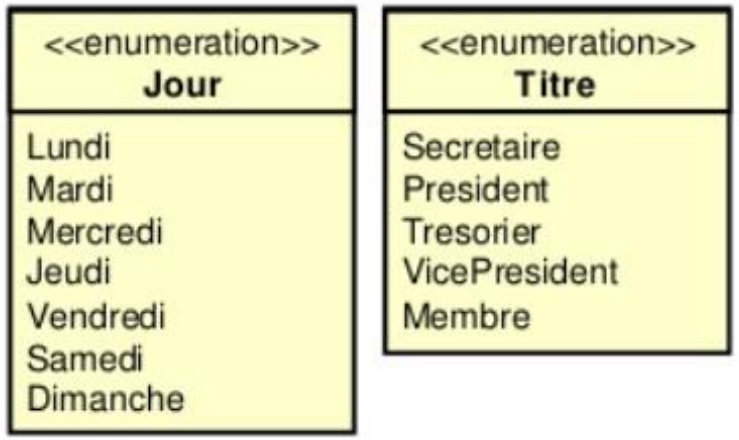
Paquetages

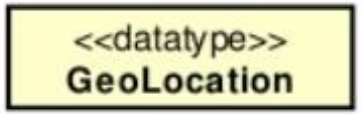
- Un paquetage (*package*) peut regrouper des classes, des interfaces et des paquetages

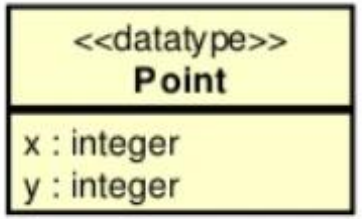


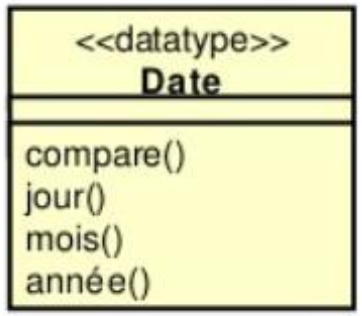
- Relations entre paquetages :
  - Relations classiques entre classes
  - Relations d'import

Enumération et types de données









- Utilisable comme type d'attributs
- Valeurs (pas d'identité)

- Type de données typiquement définis dans des « librairies »

Exemple d'utilisation

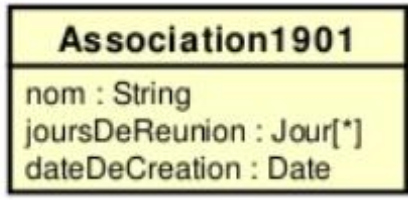


Diagramme de classes participantes

Diagramme de classes participantes

- Le diagramme de classes participantes effectue la jonction entre les cas d'utilisation, le modèle du domaine et les diagrammes de conception logicielle (ou conception détaillée).

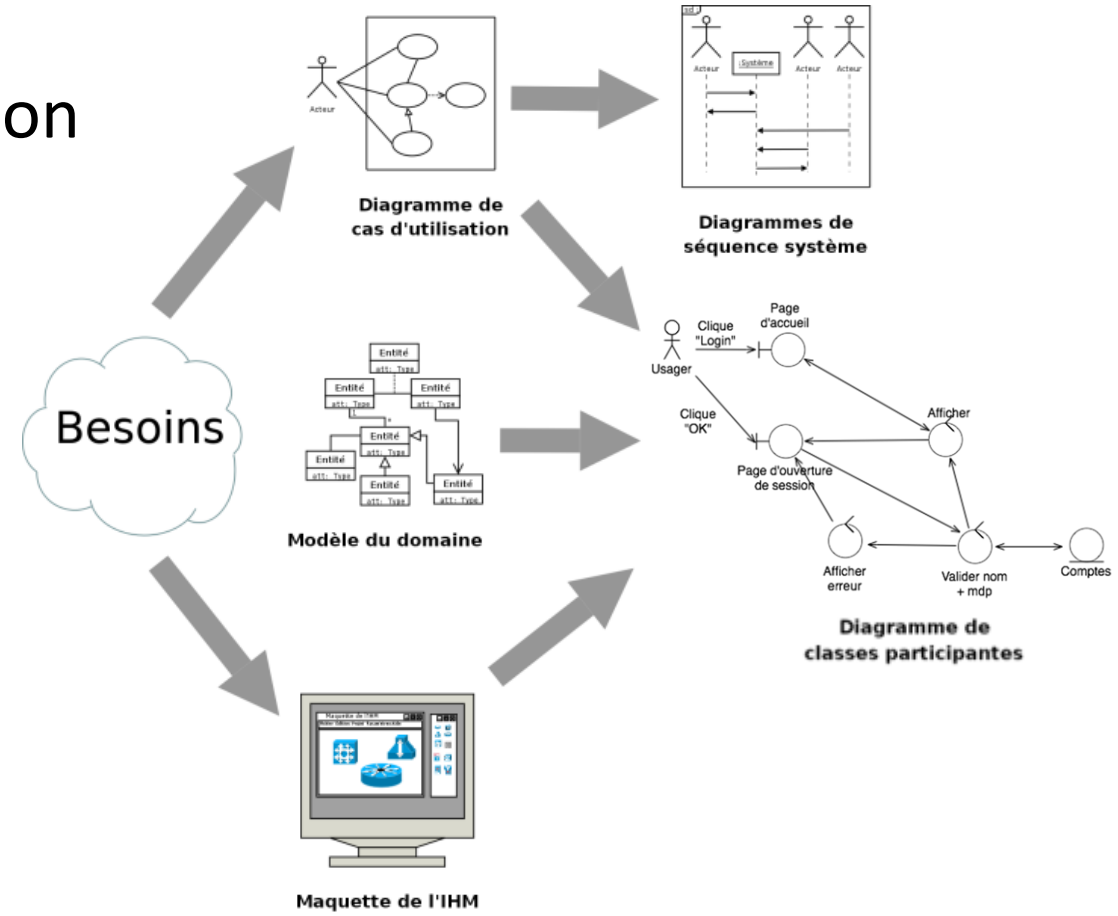


Diagramme de classes participantes

- Important pour le principe fondamental du découpage en couches d'une application :
  - Les utilisateurs ne doivent pas directement interagir avec les instances des classes du domaine par le biais de l'interface graphique.
  - Le modèle du domaine doit être indépendant des utilisateurs et de l'interface graphique.
  - L'interface graphique du logiciel doit pouvoir évoluer sans répercussion sur le cœur de l'application.
- Le diagramme de classes participantes modélise trois types de classes d'analyse, les *dialogues*, les *contrôles* et les *entités* ainsi que leurs relations.

Diagramme de classes participantes

- Les classes de dialogues :
  - permettent les interactions entre l'IHM et les utilisateurs.
  - Il y a au moins un dialogue pour chaque association entre un acteur et un cas d'utilisation du diagramme de cas d'utilisation.
- Les classes de contrôles :
  - Modélisent la cinématique de l'application.
  - Font la jonction entre les dialogues et les classes métier en permettant aux différentes vues de l'application de manipuler des informations détenues par un ou plusieurs objets métier.
  - Elles contiennent les règles applicatives et les isolent à la fois des dialogues et des entités.
- Les classes entités :
  - Proviennent directement du modèle du domaine.
  - Elles sont généralement persistantes.

Pour faire simple

- Classes de **dialogue**
  - Interactions entre le système et l'environnement/acteurs

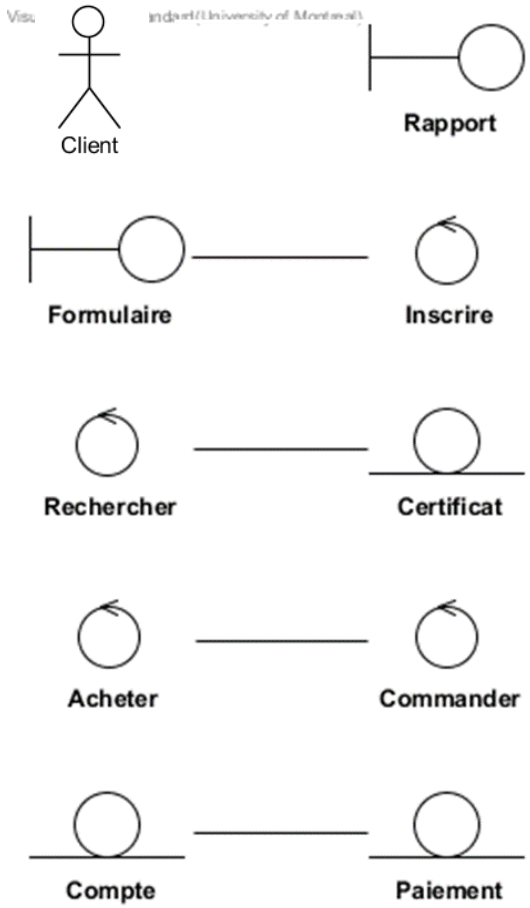
- Classes de **contrôle**
  - Calculs et algorithmes complexes

- Classes d'**entité**
  - Concepts et information qui vit et persiste dans le logiciel

Règles pratiques à suivre

- **Entités** issues du modèle du domaine ne comportent **que des attributs**
- **Contrôles ne comportent que des opérations**
  - Chaque contrôle est généralement associé à un CU et vice versa
  - Peut décomposer un CU complexe en plusieurs contrôles
- **Contrôles peuvent être associés à tous les types de classes**
  - Contrôle vers interface, vers entité ou vers autre contrôle (et l'inverse)
- **Interfaces peuvent contenir des attributs et opérations**
  - Attributs représentants des informations ou paramètres saisis par l'utilisateur ou des résultats d'actions
  - Opérations réalisent les actions que l'utilisateur demande, généralement par délégation aux contrôles

Règles du diagramme de classes participantes



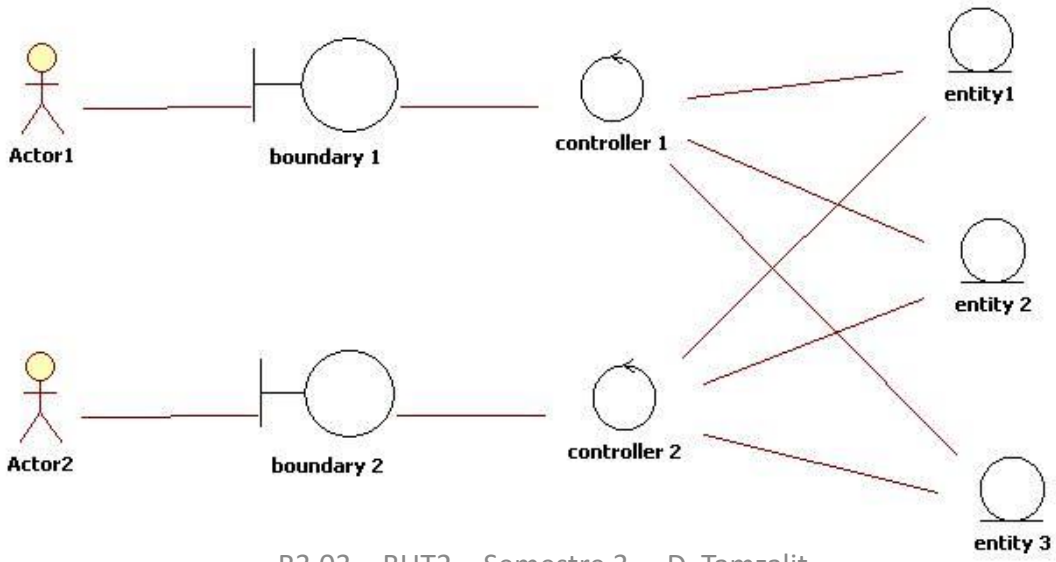
Boundary Class

Control Class

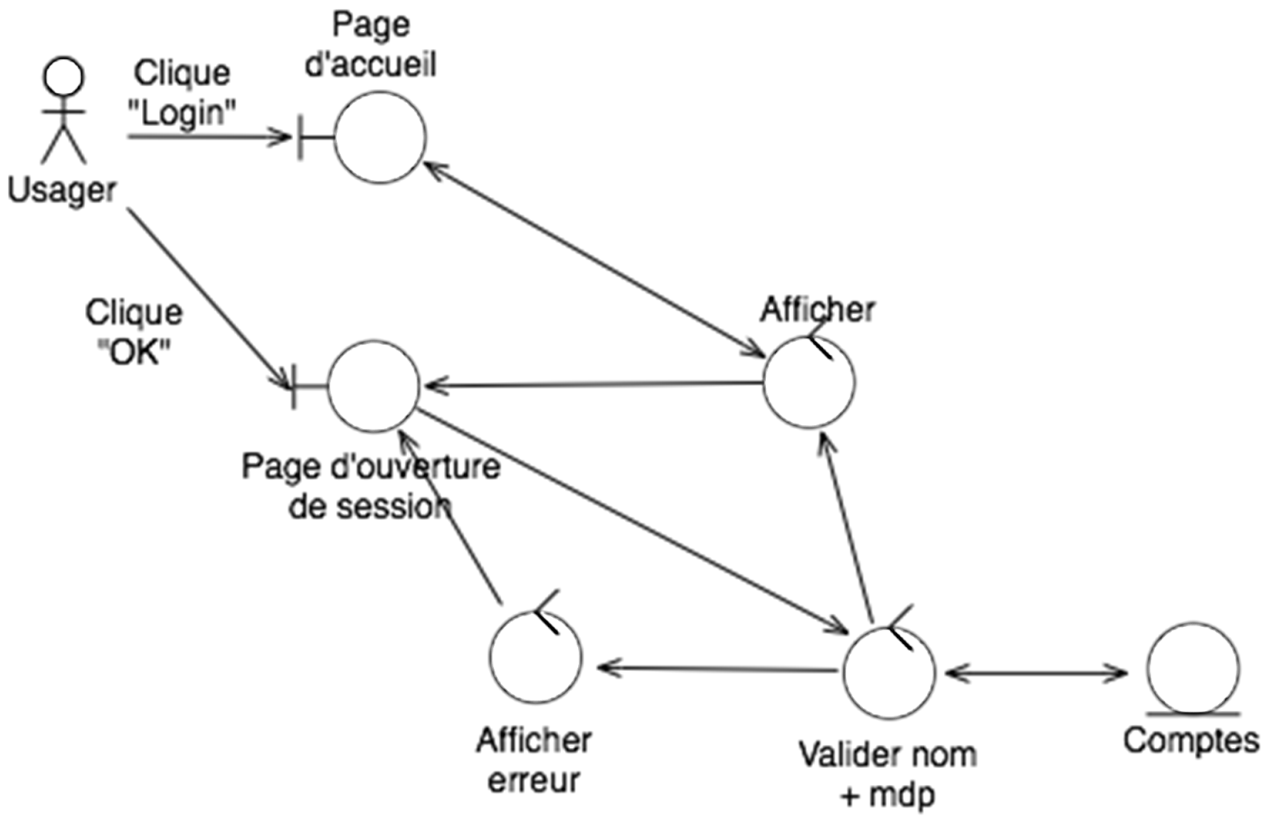
Entity Class

Patron Entité-Contrôle-Interface

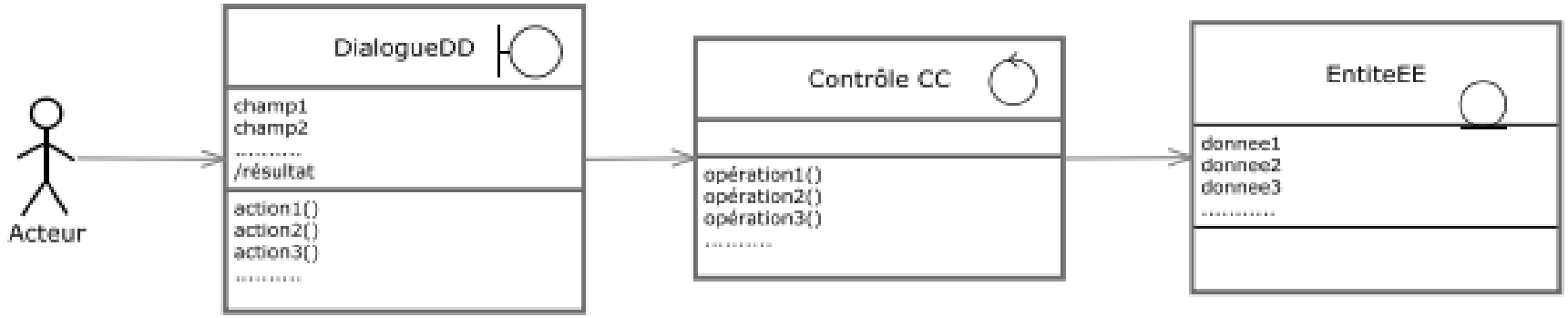
- Les acteurs n'interagissent qu'avec des classes d'interfaces
- Les entités représentent les données du système
- Les contrôles sont les médiateurs entre interfaces et entités



Exemple


Page
d'accueil
Clique
"Login"
Usager
Clique
"OK"
Afficher
Page d'ouverture
de session
Afficher
erreur
Valider nom
+ mdp
Comptes

Autre formalisme



- Ce schéma représente une implémentation de l'architecture 3-tiers, le pattern Modèle-Vue-Contrôleur (MVC).

Extraction des noms

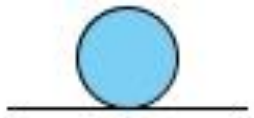
Des rapports hebdomadaires doivent être imprimés montrant combien d'argent est disponible pour les hypothèques. De plus, la liste des investissements et des hypothèques doit être imprimée sur demande.

Extraction des noms

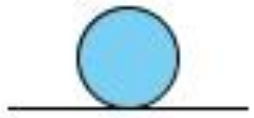
Des **rapports** hebdomadaires doivent être imprimés montrant combien d'**argent** est disponible pour les **hypothèques**. De plus, la **liste** des **investissements** et des **hypothèques** doit être imprimée sur demande.

Classes entités

- `Rapport` et `Liste` ne sont pas des informations devant persister dans le système: probablement pas des classes entités
  - Néanmoins, `Rapport` va sûrement être une classe interface
- `Argent` est un mot abstrait, trop général
- Deux candidats pour les classes d'entité



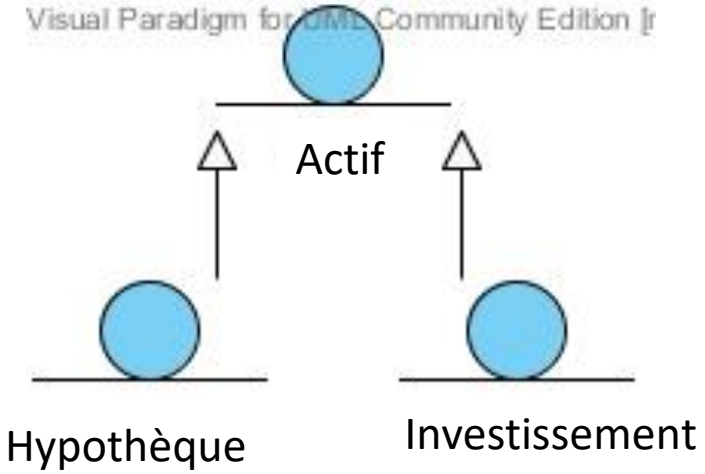
Hypothèque



Investissement

2e itération du diagramme de classes initial

- **Généraliser** les classes `Hypothèque` et `Investissement` par une classe `Actif`
  - `Actif` est la **superclasse**, ou « classe parente ».
  - `Hypothèque` et `Investissement` sont des **sous-classes** d'`Actif`



Ressources

- Précédents cours DUT
- UML2 par la pratique, Pascal Roques, Editions Eyrolles,
- UML2, de l'apprentissage à la pratique, Laurent Audibert, Edition Ellipses, https://laurent-audibert.developpez.com/Cours-UML/?page=mise-en-oeuvre-uml .
- Support cours E. Syriani, Université de Montréal.