Ingénierie des Modèles

Modélisation logicielle

Léa Brunschwig
✉ lea.brunschwig@univ-pau.fr

M2 Technologies de l'Internet

Université de Pau et des Pays de l'Adour
Collège STEE
Département Informatique

01.
Concepts principaux
Modèle, méta-modèle, transformation.

02.
Modélisation logicielle
UML et OCL.

03.
Méta-modélisation et DSL
Syntaxe concrète et abstraite, …

04.
Transformation de modèles
M2M, M2T/M2C, Acceleo, …

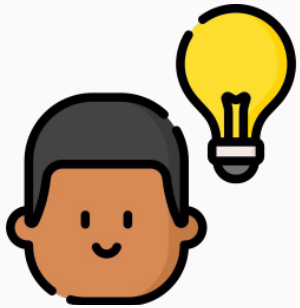
Rôle de la modélisation logicielle

Communication

- langage **commun** entre les ≠ parties prenantes d'un projet logiciel,
  - développeurs, concepteurs, gestionnaires de projet, clients, …
- rend les concepts et exigences plus **compréhensibles** et **accessibles à tous**.

Compréhension du système

- permet aux développeurs de mieux **comprendre** le système logiciel en terme de :
  - structure,
  - comportement,
  - interactions.
- aide à **clarifier** les besoins et objectifs du projet.



Rôle de la modélisation logicielle

Documentation visuelle

- **enregistre** les décisions de conception, spécifications fonctionnelles et contraintes du système,
- **facilite** la **maintenance** ultérieure, les mises à jour et le débogage.

Analyse et conception

- identifier et **résoudre des problèmes** de conception potentiels avant l'implémentation,
- **vérifier** que le modèle répond aux contraintes et aux exigences du système,
    - utilisation d'outils d'analyse.

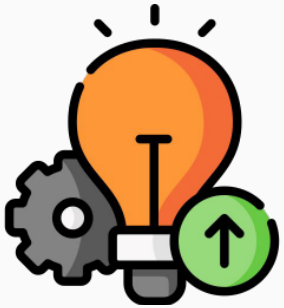
Rôle de la modélisation logicielle

Réutilisation

- modèles de conception génériques ou bibliothèques de composants réutilisables pour **accélérer le développement** de nouveaux projets,
  - Design Patterns.

Évolution et Gestion du Changement

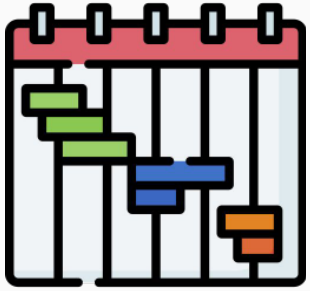
- permet de **gérer** efficacement les **évolutions**, mise à jour et extensions de logiciel,
- reflète les **changements** dans le système au fil du temps.

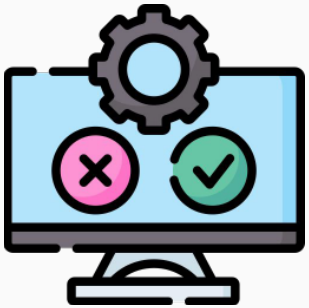


Rôle de la modélisation logicielle

Gestion de projet

- aide à **planifier** et **suivre** le développement du logiciel,
- permet d'**estimer** les ressources nécessaires, identifier les dépendances, …



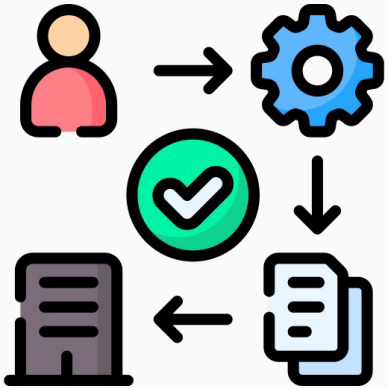


Test et Validation

- sert de **base** pour la création de **test**,
  - scénarios de tests peuvent être dérivés de diagrammes de séquences et d'états,
- facilite la **validation** du système.

Rôle de la modélisation logicielle

- représenter **visuellement**, de manière **abstraite** les différents aspects d'un **système logiciel**,
- offre une méthode **structurée** pour les systèmes logiciels afin de les :
  - **comprendre**,
  - **concevoir**,
  - **documenter**
  - **gérer**.
- favorise une meilleure **communication**, prise de **décision** et **réduction des erreurs** de conception ainsi qu'une **gestion** plus efficace des projets.



Object Management Group (OMG)

Source : https://www.omg.org

- Organisation internationale à but non lucratif fondé en 1989
- Oeuvre pour le développement et la promotion de normes dans l'industrie informatique pour favoriser :
  - l'**interopérabilité**,
  - la **réutilisation**,
  - la **normalisation**.

- Exemple de contributions OMG :
  - Unified Modeling Language (**UML**),
  - Common Object Request Broker Architecture (**CORBA**),
  - Model-Driven Architecture (**MDA**).

Unified Modeling Language (UML)

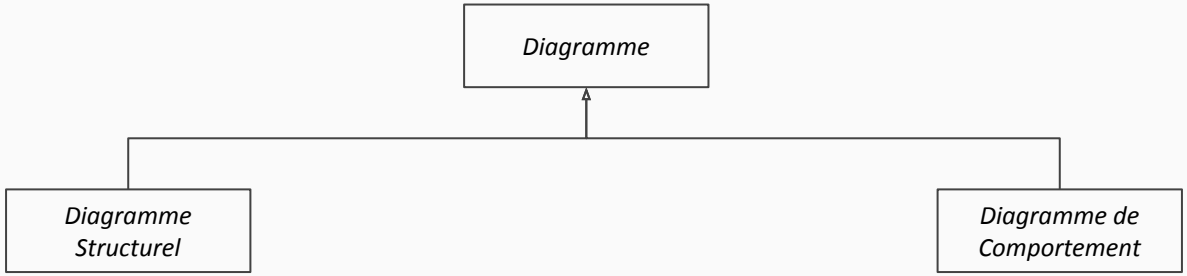
Source : https://www.uml.org/

- langage de modélisation **visuelle**,
- utilisé dans le domaine du développement logiciel et l'ingénierie des systèmes,
- représente graphiquement des systèmes logiciels et processus,
- fournit un ensemble de **notations standardisées** sous forme de diagramme.

❌ **UML ≠ processus de développement** → n'établit pas des modèles précis à utiliser

Diagrammes UML

- 14 diagrammes différents :



Structurel

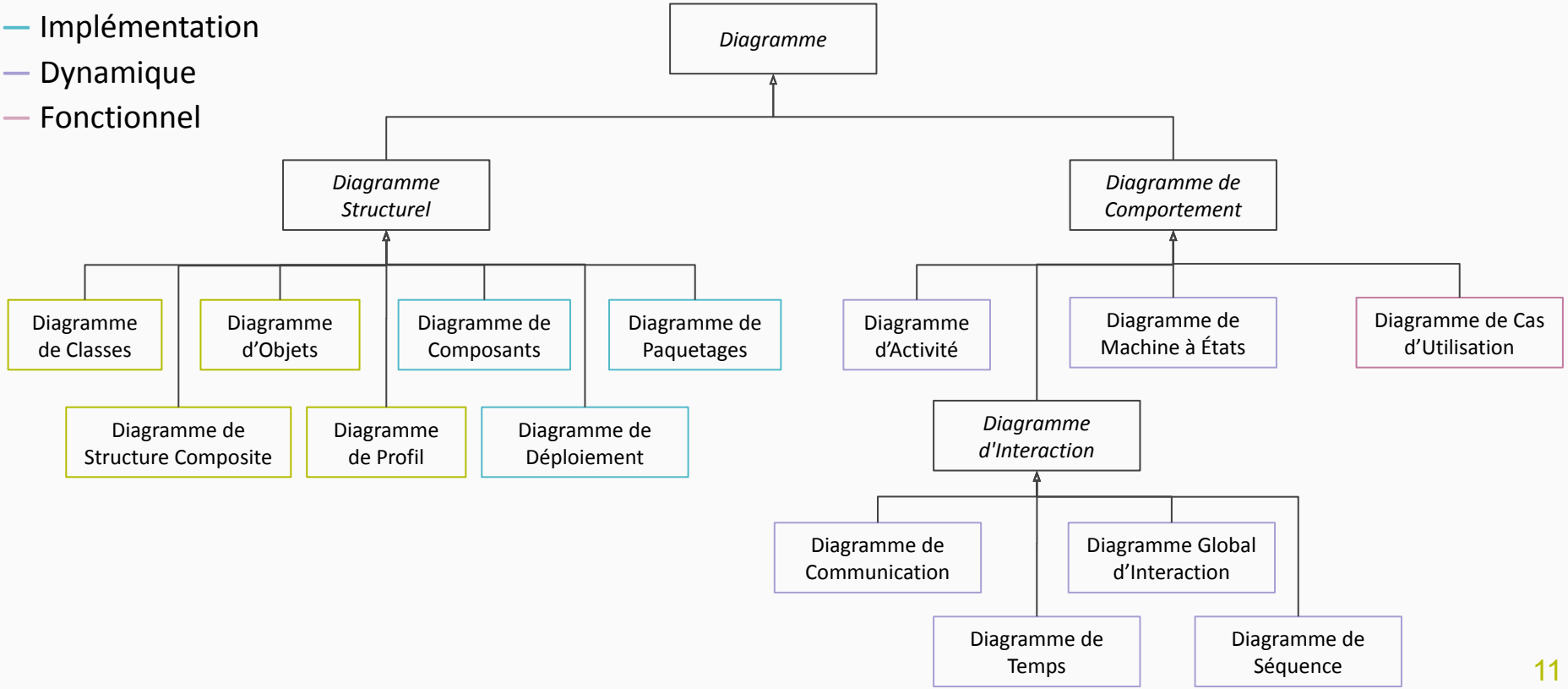
Décrit les éléments du système et ses relations.

Comportement

Décrit les comportements du système au fil du temps.

Diagrammes UML

- Statique
- Implémentation
- Dynamique
- Fonctionnel



Modélisation UML

- Permet de décrire différents aspects d'un même système.

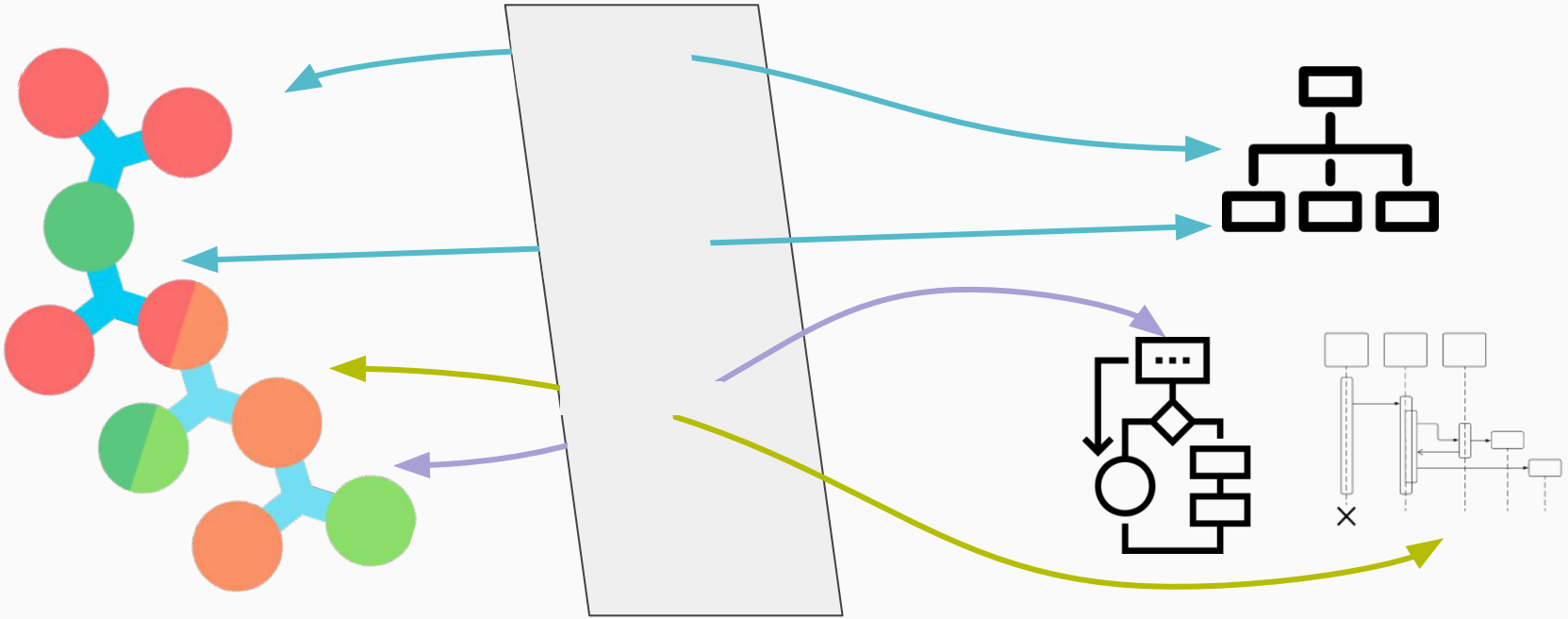


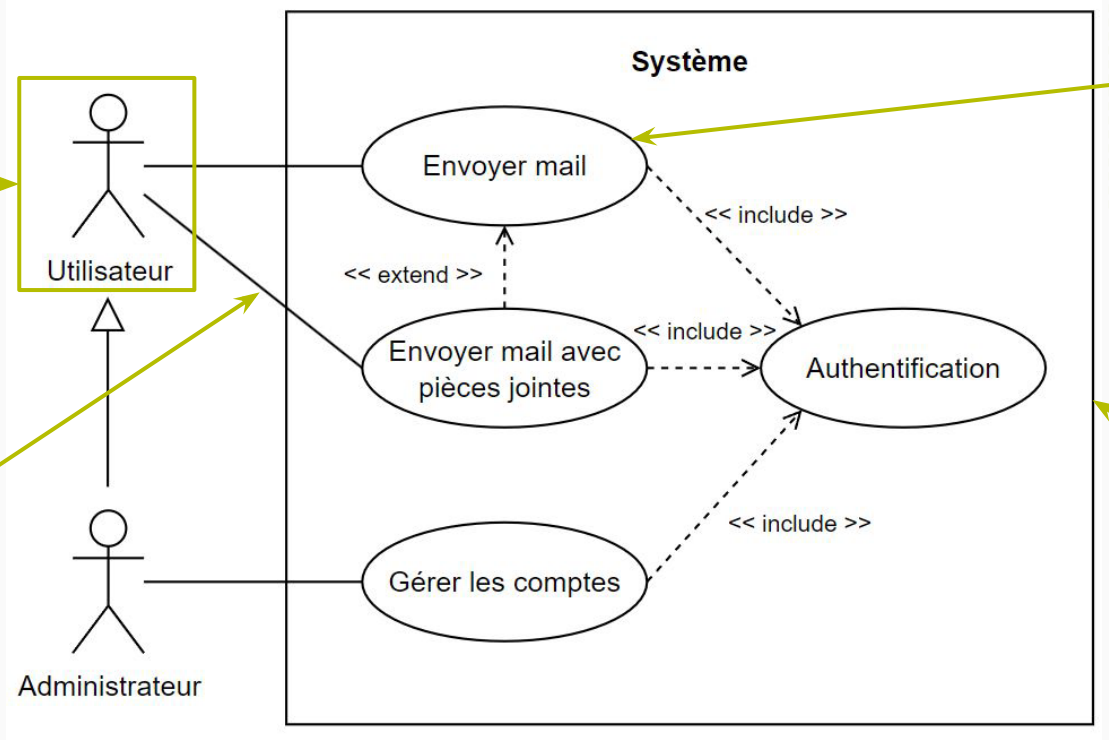
Diagramme de Cas d’Utilisation

Diagramme de Cas d'Utilisation

- Use Case Diagram
- Montre comment les **utilisateurs interagissent** avec un **système** pour accomplir des **actions** ou des **scénarios** spécifiques.

Acteur
Personne externe, processus ou chose qui interagit avec un système.

Association
Communication entre un acteur et un cas d'utilisation.



Cas d'utilisation
Unité cohérente représentant une fonctionnalité visible de l'extérieur.

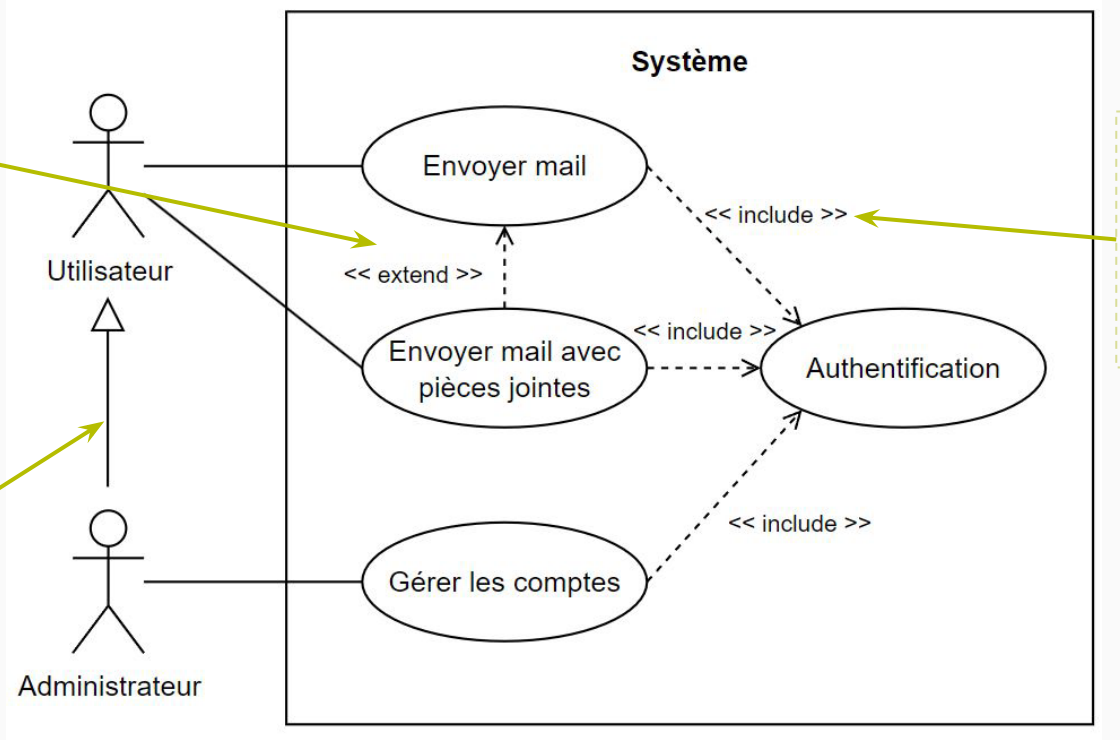
Limites du système

Diagramme de Cas d'Utilisation

- Use Case Diagram
- Montre comment les **utilisateurs interagissent** avec un **système** pour accomplir des **actions** ou des **scénarios** spécifiques.

Extension
A étend **B** lorsque **A** peut être appelé au cours de l'exécution de **B**.

Généralisation
A est une généralisation de **B** si **B** est un cas particulier de **A**.



Inclusion
A inclut un cas **B** si le comportement décrit par le cas **A** inclut le comportement du cas **B**.

Diagramme de Cas d'Utilisation

- Pour chaque cas d'utilisation description textuelle de son but et du rôle des utilisateurs.

Nom : *Utiliser une tournure à l'infinitif*

Envoyer un mail.

Objectif : *Description résumée permettant de comprendre l'intention principale*

Processus par lequel un utilisateur peut rédiger et envoyer un mail via le système.

Acteurs principaux : *Ceux qui vont réaliser le cas d'utilisation*

Utilisateur

Acteurs secondaires : *Ceux qui ne font que recevoir des infos*

Aucun

Dates : *Dates de création et de mise à jour de la description courante*

12/09/2023

Responsable : *Noms des responsables*

Léa Brunschwig

Version : *Numéro de version*

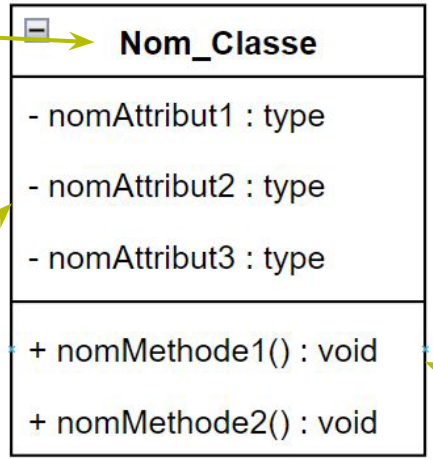
1.0

Diagramme de Classes et d’Objets

Diagramme de Classes

- Class Diagram
- Définit les éléments formant une application et leurs relations.

Nom de la Classe
Débute avec une lettre majuscule, centré, écrit en **gras**, *italique* si abstraite.

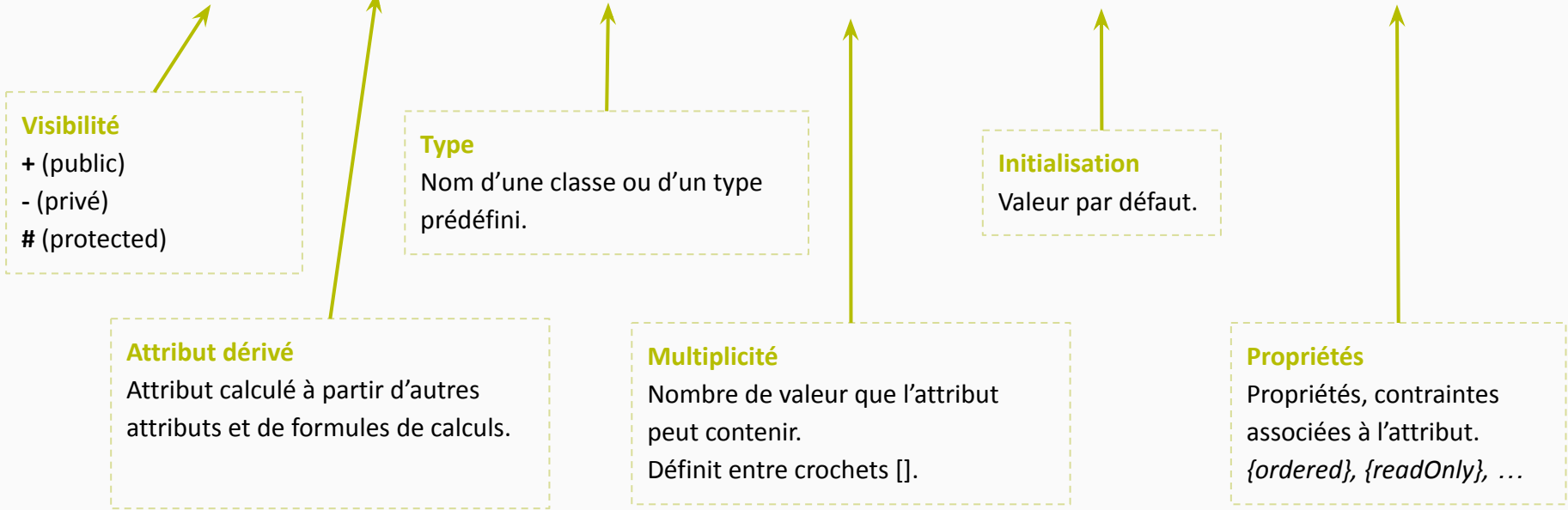


Attributs
Élément caractérisant une partie de l'état d'un objet.
`visibilité nom : type [multiplicité] = init {propriétés}`

Opérations
Méthodes qu'une classe sait exécuter.
`visibilité nom(paramètres) : typeRetour {propriétés}`

Diagramme de Classes : Attributs

`[visibilité] [/] nom [: type] [multiplicité] [= init] [{propriétés}]`



- Si souligné → attribut statique.

Diagramme de Classes : Opérations

```
[visibilité] nom ([param_1, … param_N]) [: typeRetour] [{propriétés}]
```

Visibilité
+ (public)
- (privé)
(protected)

Type de retour
Nom d'une classe ou d'un type prédéfini.
Void si aucun.

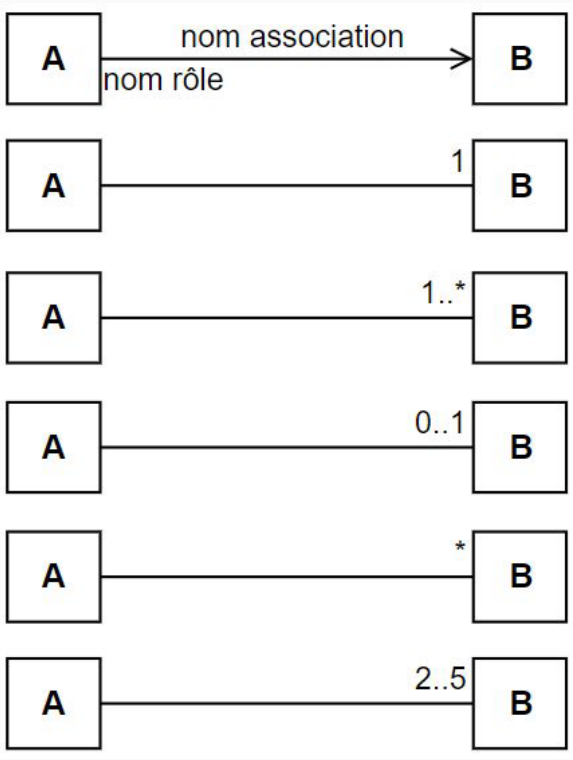
Propriétés
Propriétés, contraintes associées à la méthode.
{abstract}, …

```
[direction] nom : type [multiplicité] [= valeurDefaut]
```

Direction
in : paramètre d'entrée
out : paramètre de sortie
inout : entrée/sortie

- Si souligné → méthode statique.

Diagramme de Classes : Relations



Association unidirectionnelle

Une instance de **A** est toujours associée avec **une** instance de **B**

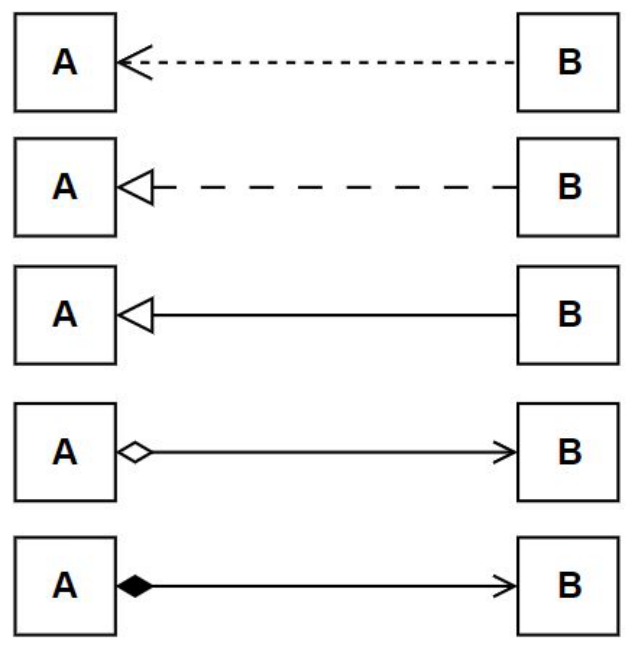
Une instance de **A** est toujours associée avec **une ou plusieurs** instances de **B**

Une instance de **A** est associée avec **zéro ou une** instance de **B**

Une instance de **A** est associée avec **zéro, une ou plusieurs** instances de **B**

Une instance de **A** est associée avec **entre deux et cinq** instances de **B**

Diagramme de Classes : Relations



Dépendance : **A** utilise **B** de manière ponctuelle.
ex.: Client passe une Commande.

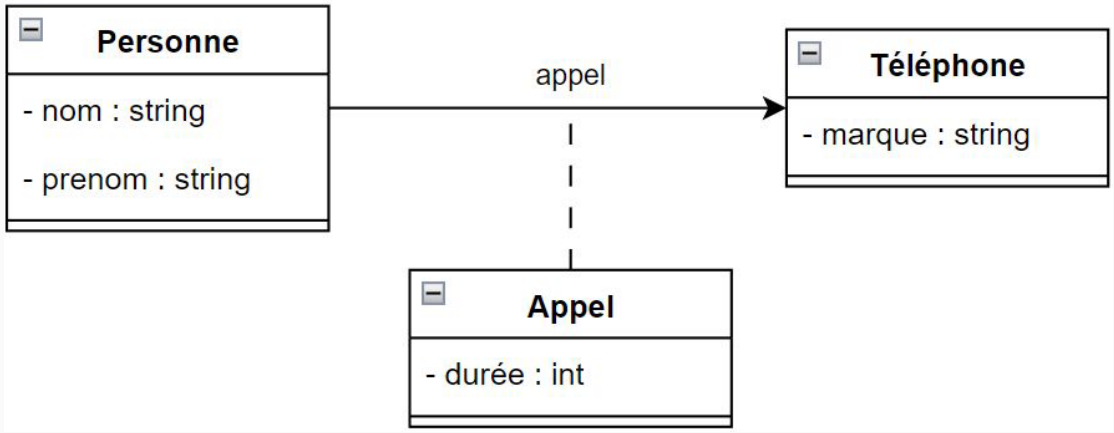
Implémentation : relation entre une classe **B** et interface **A**

Héritage : **B** est une spécialisation de **A**
ex.: Chien hérite d'Animal.

Composition : **A** est constitué d'un objet **B**. Si **A** disparaît, **B** disparaît aussi.
ex. : Chien est composé d'un Cerveau et d'un Coeur.

Aggregation : **A** possède un autre objet **B** qui existe indépendamment.
ex. : Chien a un Collier.

Diagramme de Classes : Classe association



- Apporte des informations qui n'appartiennent pas aux deux autres classes.

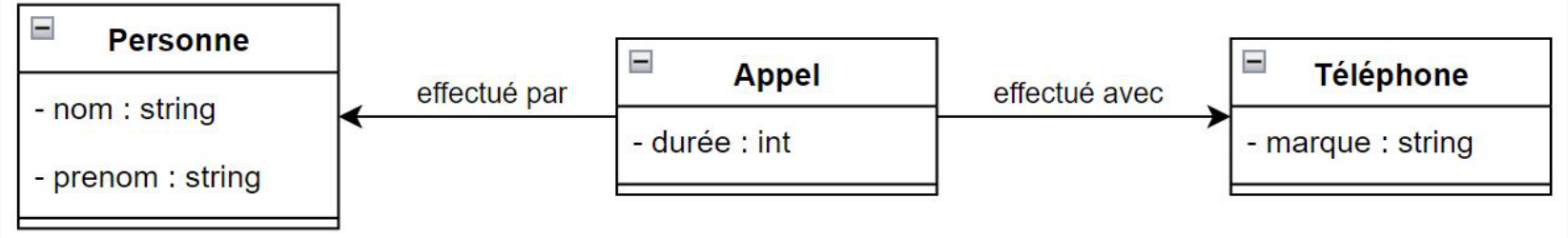
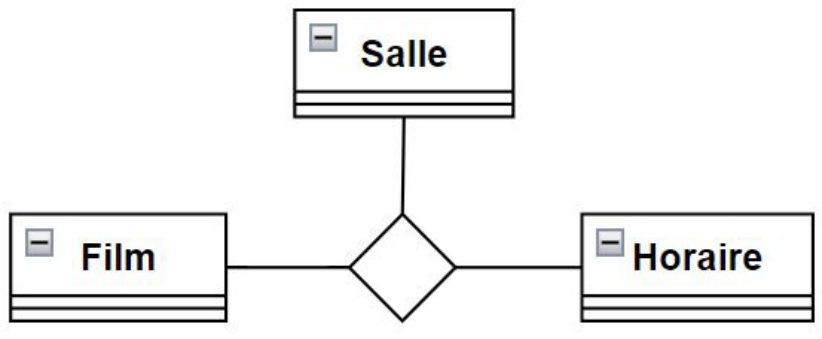


Diagramme de Classes : Association n-aire



- Lie plus de deux classes entre elles.

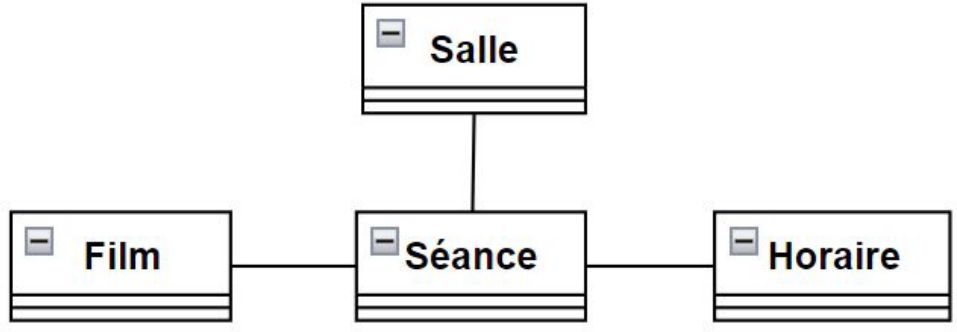
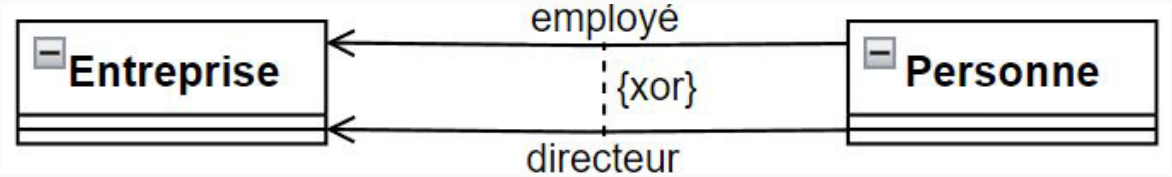


Diagramme de Classes : Contraintes sur les associations

- **Relation d'exclusion** : soit l'une soit l'autre mais pas les deux à la fois



- **Subset** : Une association peut-être le sous-ensemble d'une autre

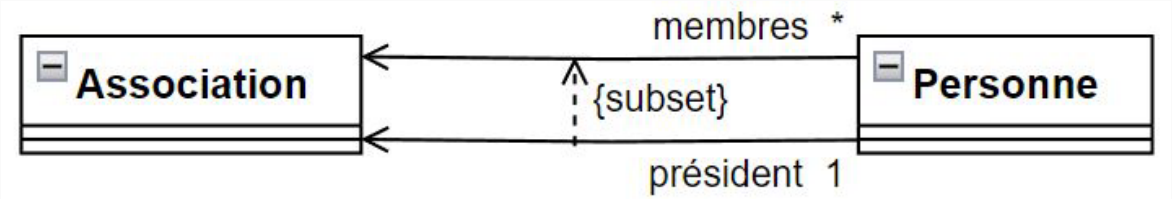


Diagramme d'Objets

- Object Diagram
- Représente les objets d'un système et leurs relations à un instant donné,
- A un diagramme de classe correspond un **infinité** de diagramme d'objet.

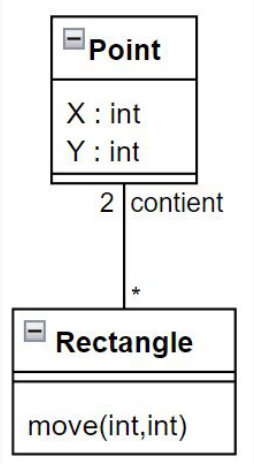


Diagramme de Classes

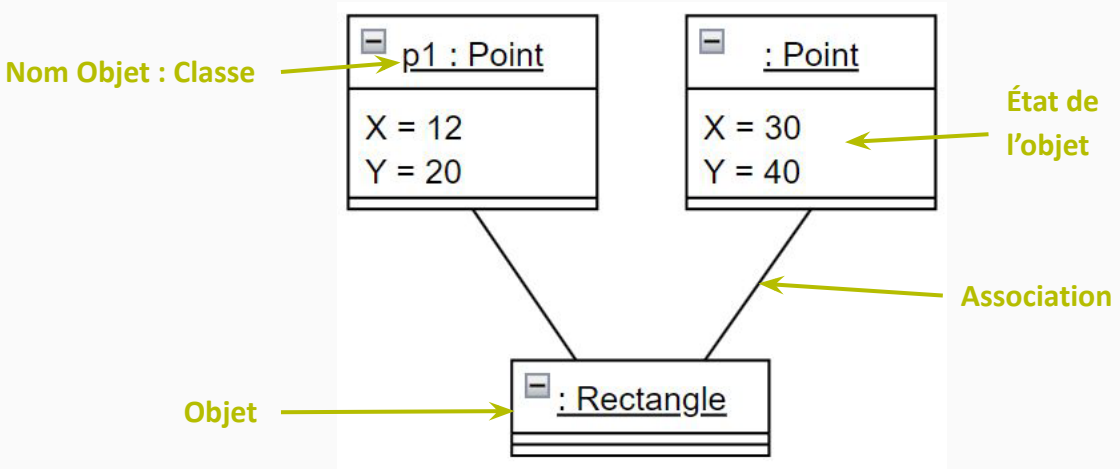


Diagramme d'Objets

Diagramme d’Implémentation

Diagramme de composants

- Component Diagram
- Décrit l'organisation du système du point de vue des éléments logiciels,
- Ensemble de composants connectés entre eux par assemblage ou composition.

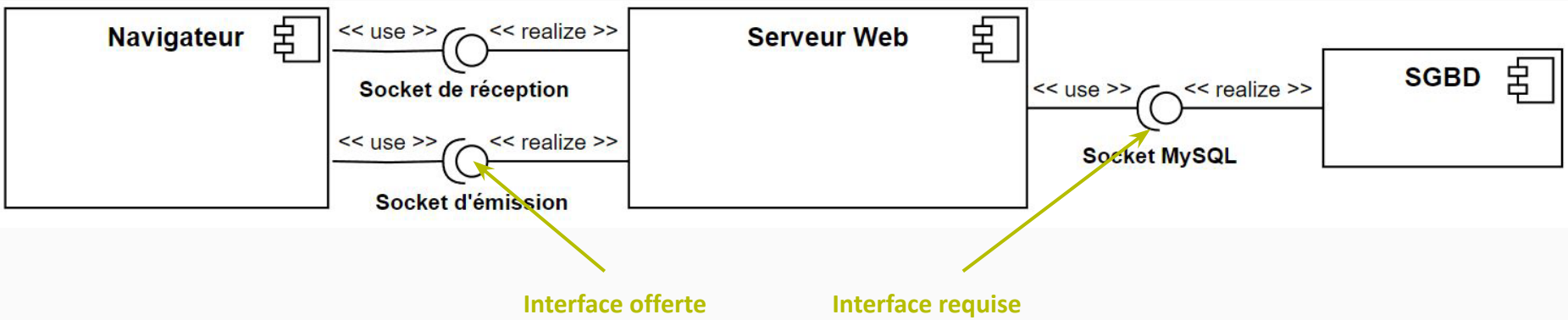


Diagramme de paquetage

- Package Diagram
- Décrit l'organisation et les relations entre les paquetages composant un système.

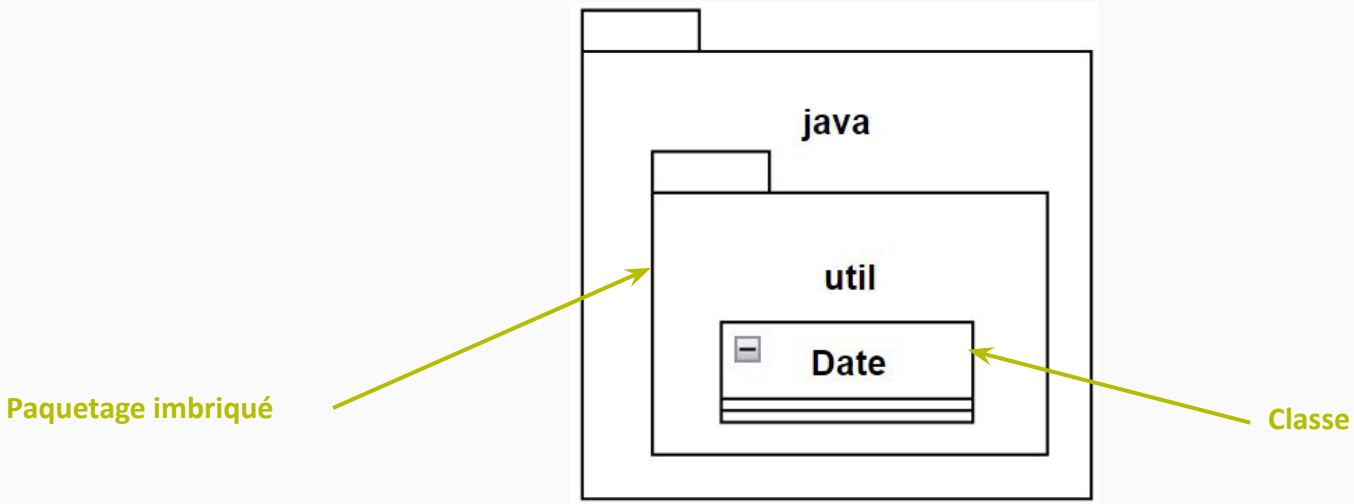


Diagramme d’Activité

Diagramme d'Activité

- Activity Diagram
- Permet de comprendre un flot de données et d'analyser des cas d'utilisation.

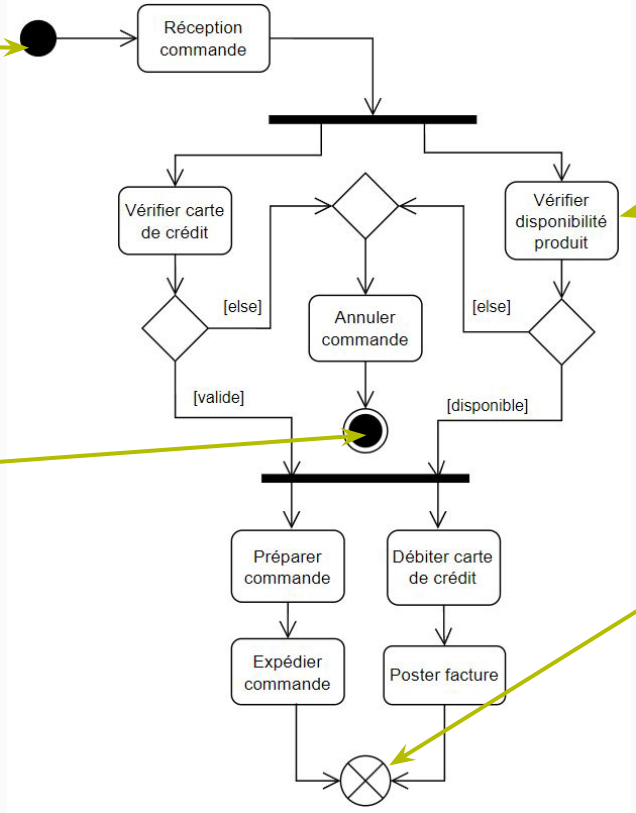
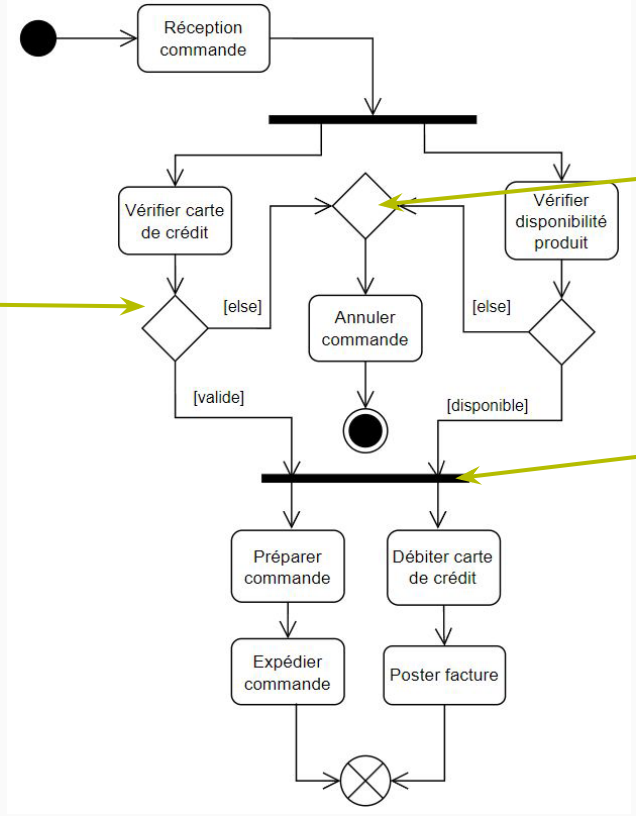


Diagramme d'Activité

- Activity Diagram
- Permet de comprendre un flot de données et d'analyser des cas d'utilisation.



Noeud de fusion

Noeud de décision

Noeud de jointure/bifurcation

Diagramme de Séquence

Diagramme de Séquence

- Sequence Diagram
- Spécifie l'ordonnancement temporel des interactions entre les objets.

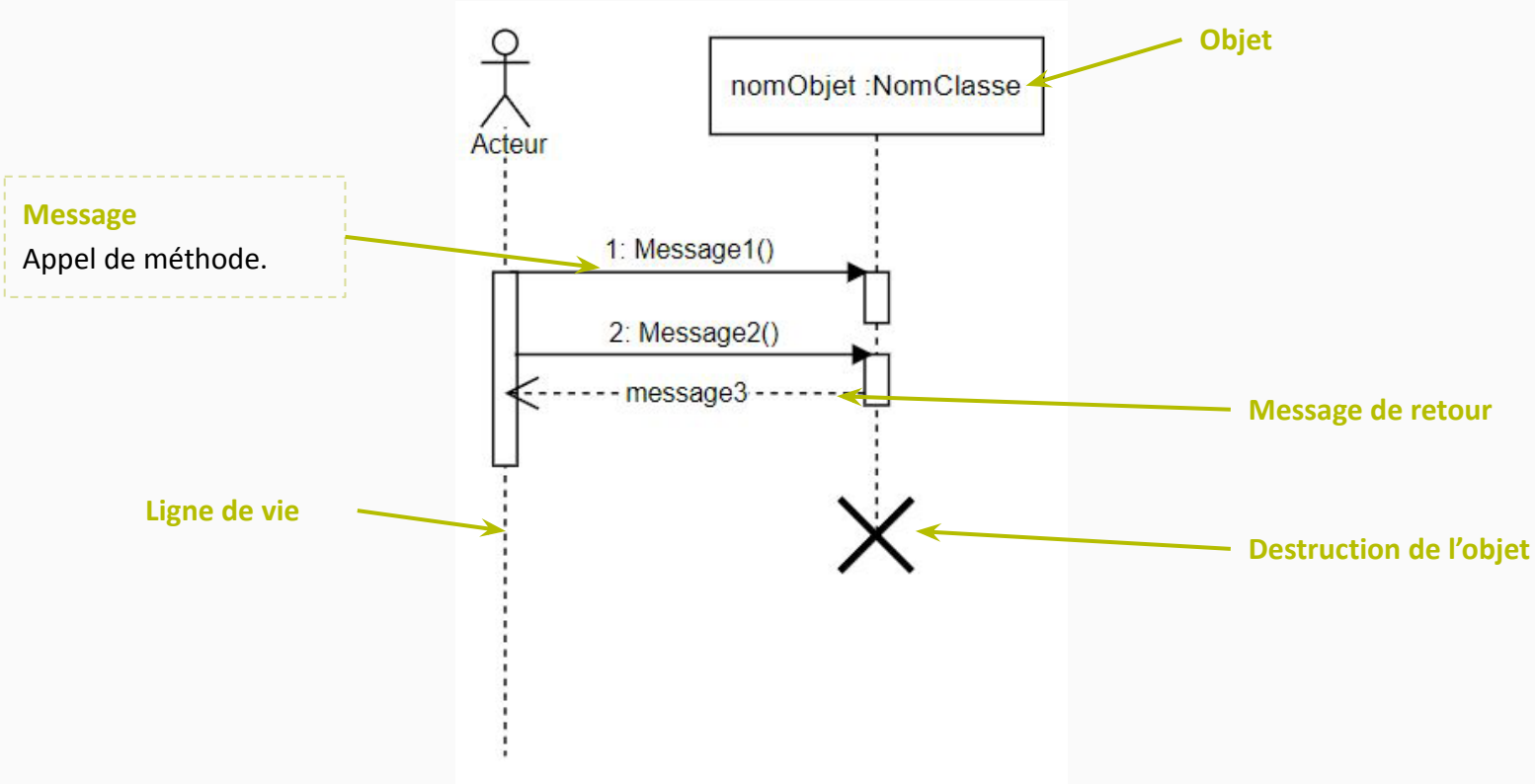


Diagramme de Machines à États

Diagramme de Machines à États

- State Machine Diagram
- Comportement interne d'un objet

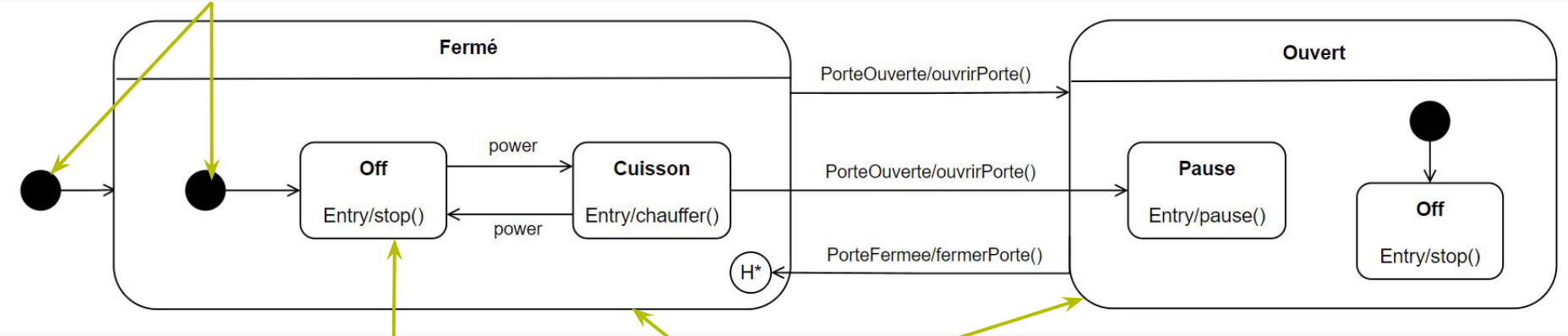
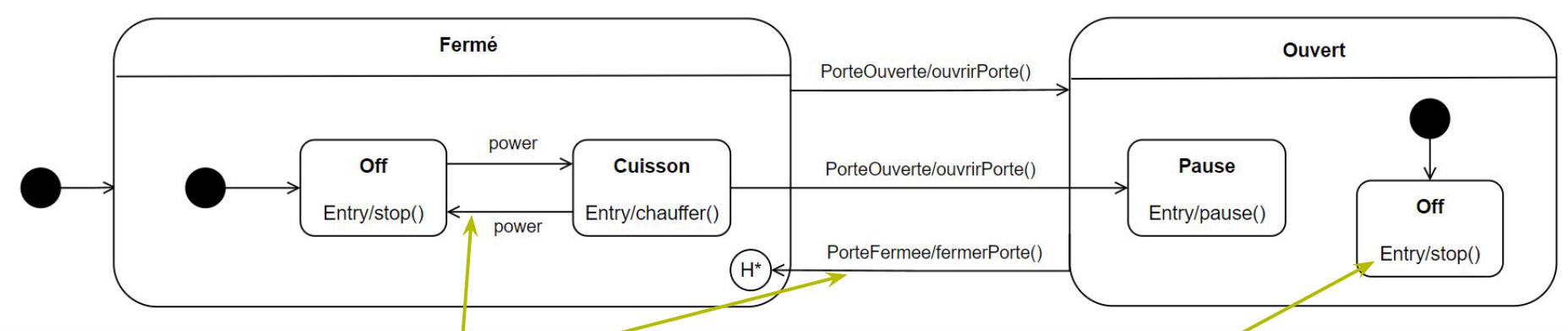


Diagramme de Machines à États

- State Machine Diagram
- Comportement interne d'un objet



Transition

`[guarde] évenement [/ operation()]`

A un événement déclencheur, une source, une cible.

Guarde : condition bouléenne.

Opération : comportement associé.

Activité

Associée à un état.

`do/action` action exécutée dans l'état

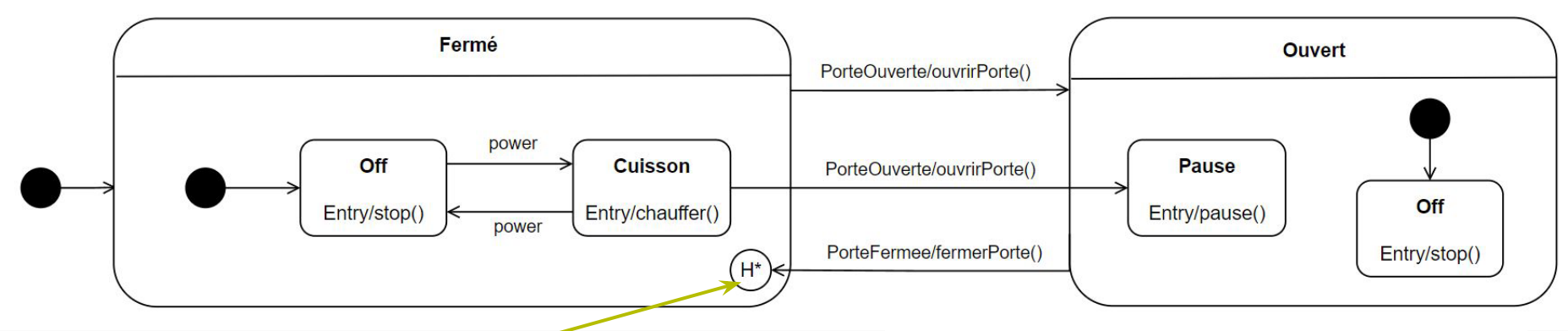
`entry/action` action exécutée à l'entrée dans l'état

`exit/action` action exécutée à la sortie de l'état

`evt/action` transition interne dès que `evt`

Diagramme de Machines à États

- State Machine Diagram
- Comportement interne d'un objet

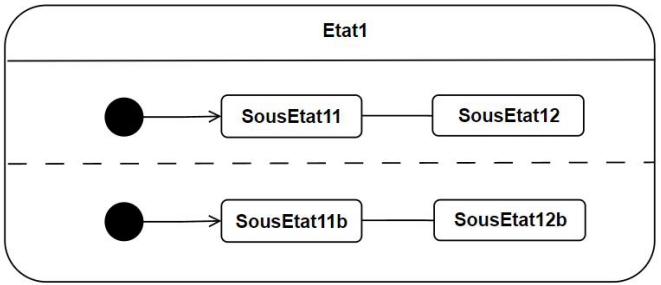


Historique

Permet de revenir dans l'état interne qui était celui avant que l'on quitte

`H*` *Deep History* réactive le dernier état interne de tous les états composites

`H` *Shallow History* réactive le premier niveau



Parallélisme