جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université Abderrahmane Mira de Bejaïa
Faculté des sciences exactes
DEPARTEMENT D'INFORMATIQUE

# *Génie Logiciel*

**Niveau : 3ème année licence (MI)**

## Chapitre 2 : Modélisation avec UML

**Chargés de cours :**
**Dr M. MOHAMMEDI**
**& Dr N. YESSAD**

**Année universitaire 2024/2025**

# Plan du chapitre 2

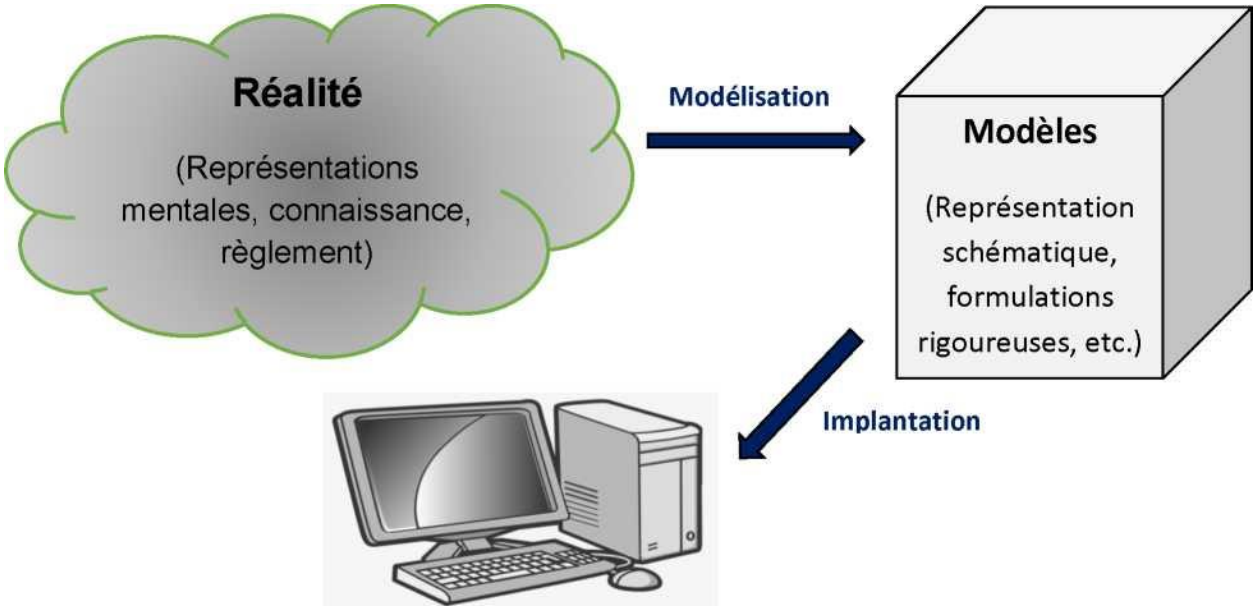
- Introduction
- Exploration intégrée: Modélisation, modèle, conception orientée objet et évolutions d'UML
- Les diagrammes UML
- Mécanismes généraux
- Paquetages

# Introduction

- **UML (Unified Modeling Language)** est un langage standardisé utilisé dans le domaine de l'**ingénierie logicielle orientée objet**;
- Il est **composé** d'un ensemble de techniques de **notation graphique** qui permettent de **créer** des **modèles visuels** de systèmes logiciels **orientés objet**;
- **UML intègre** des techniques de **modélisation de données**, de **modélisation d'entreprise**, de **modélisation d'objets** et de **modélisation de composants**;
- Il peut être **utilisé** à différentes **étapes** du cycle de vie du **développement logiciel**;
- **UML** est **compatible** avec différentes **technologies** de **mise en œuvre**.

# Modélisation

Une **modélisation** est une représentation **simplifiée** d'une **réalité**. Elle permet de **capturer** des aspects **pertinents** pour répondre a un **objectif défini a priori**.




Exemple type d'une modelisation.

**Exemple :**

Un **astronaute** modélisera la **Lune** comme un corps **céleste** ayant une certaine **masse** et se trouvant a une certaine **distance** de la **Terre**, alors qu'un **poète** la modélisera comme une **dame** avec qu'il peut **avoir** une **conversation**.

# Modèle

- Un **modèle** est une **représentation abstraite** de la **réalité** qui **simplifie** un **phénomène** en **excluant** certains **détails** du monde réel.
- Il vise à **réduire** la complexité en **éliminant** les détails **insignifiants** pour le comportement du **phénomène étudié**.
- Le **modèle** reflète ce que son **concepteur** considère comme **important** pour la **compréhension** et la **prédiction** du **phénomène**.
- Les limites du **modèle** dépendent des **objectifs fixés** par le **concepteur**.
- Quand un **modèle** devient **complexe**, il est préférable de le **décomposer** en plusieurs **modèles** simples et **manipulables**.
- L'expression d'un **modèle** se fait dans un **langage compatible** avec le **système étudié** et les **objectifs visés**.

# Modèle (suite)

- Par exemple, un **physicien** utilisera les **mathématiques** comme langage de **modélisation** pour représenter la **Lune**.

Dans le domaine du **développement logiciel**, l'un des **langages** utilisés pour la modélisation est l'**Unified Modeling Language** (**UML**). Il possède sa propre **sémantique** et une **syntaxe** constituée de **graphiques** et de **texte**, permettant de **représenter** des **concepts** sous **différentes formes**, notamment à travers l'utilisation de **diagrammes**.

# Modélisation Orientée Objet

La Conception Orientée Objet (**COO**) est une **méthode** de **conception** qui permet de créer des **architectures logicielles** basées sur les **objets du système**, plutôt que sur une **décomposition fonctionnelle**. La **COO** se **caractérise** par les **éléments** suivants :

- **Regroupement** des **données** et des **traitements**.
- **Réduction** de l'écart entre le monde réel et sa **représentation informatique**.
- **Attribution** des responsabilités à des **emplacements spécifiques**.
- **Décomposition** basée sur l'identification des **relations** entre les **objets**.

# UML en Application

Au milieu des années **1990**, les auteurs de **Booch**, **OOSE** (Object Oriented Software Engineering) et **OMT** (Object Modeling Technique) ont décidé de **créer** un **langage** de modélisation **unifié** avec les **objectifs** suivants :

- Modéliser un **système des concepts** a **l'exécutable**, en utilisant les **techniques orienté objet**.
- **Réduire** la **complexité** de la **modélisation**.
- Être **utilisable** à la fois par les **humains** et les **machines** :
  - **Fournir** des représentations **graphiques** avec une qualité **semi-formelle** suffisante pour être **automatiquement** traduites en **code source**.

# UML en Application (suite)

- Cependant, ces représentations **ne possèdent** pas une qualité **formelle suffisante** pour justifier des propriétés **mathématiques** aussi **rigoureuses** que celles des langages de **spécification formelle** tels que Z, VDM, etc.

- **Officiellement**, **UML** a été créé en **1994**.

Il existe des langages a differents niveaux de formalisation, a savoir : (i) **langages formels** (ex. Z, B, VDM) : le plus souvent mathematiques, au grand pouvoir d'expression et permettant des preuves formelles sur les specifications ; (ii) **langages semi-formels** (ex. MERISE, UML, Orienté but, etc.) : le plus souvent graphiques, au pouvoir d'expression moindre mais plus faciles d'emploi.

# Historique d'UML

Les **grandes étapes** de la diffusion d'**UML** peuvent se resumer comme suit :

**1994 - 1996** : rapprochement des methodes **OMT**, **BOOCH** et **OOSE** et naissance de la **premiere version d'UML**.

**23 novembre 1997** : version 1.1 d'UML adoptee par l'OMG.

**1998 - 1999** : sortie des versions 1.2 a 1.3 d'UML.

**2000 - 2001** : sortie des dernieres versions suivantes 1.x.

**2002 - 2003** : preparation de la version 2.

**10 octobre 2004** : sortie de la version 2.1.

**5 fevrier 2007** : sortie de la version 2.1.1.

**Decembre 2017** : sortie de la version 2.5.1.

# Les étiquettes

- **Les étiquettes**, également connues sous le nom de **valeurs marquées** (tagged values), sont des **mécanismes** utilisés en **modélisation** pour ajouter des **propriétés supplémentaires** à un élément de modélisation **spécifique**.

- **Les étiquettes** sont représentées sous la forme **d'une paire** (**nom**, **valeur**), où le **nom** représente l'**identifiant** de la **propriété** et la **valeur** correspond à l'**information associée**.

L'objectif **fondamental** des **étiquettes** est **d'enrichir** la **spécification** d'un élément de **modélisation** en lui ajoutant de **nouvelles informations**. Elles permettent de créer et d'associer des détails **spécifiques** à l'élément, qui **ne sont pas directement** définis par les **caractéristiques de base** du modèle.

# Les stéréotypes

- **Les stéréotypes** sont des **mécanismes d'annotation** appliqués à un élément de modèle pour **caractériser** et différencier **différentes variations** d'un même **concept**.

- **Les stéréotypes** ne sont pas définis **formellement**, mais ils permettent **d'adapter** le langage de **modélisation** à des **situations spécifiques**.

Un **stéréotype** est **représenté** par une **chaîne de caractères** placée entre **guillemets**, connue sous le nom de "**nom du stéréotype**". Cette **annotation** est associée au **symbole** de l'élément de **modèle de base** ou se trouve à **proximité** de celui-ci.

# Les notes

- Une **note** est un **commentaire explicatif** qui est associé à un ou **plusieurs** éléments de **modélisation**.

- Elle est **généralement** représentée **symboliquement** par un **rectangle plié à l'angle supérieur droit**.

- **Contrairement** aux autres éléments de **modélisation**, une note ne **spécifie** pas directement le type d'élément qu'elle **accompagne**. Toute **la compréhensibilité** de la note repose **entièrement** sur le **contenu de son texte**.

- Pour **relier** une note à l'élément qu'elle **décrit**, il est possible **d'utiliser** une **ligne en pointillés**.

**Notation graphique et exemple :**

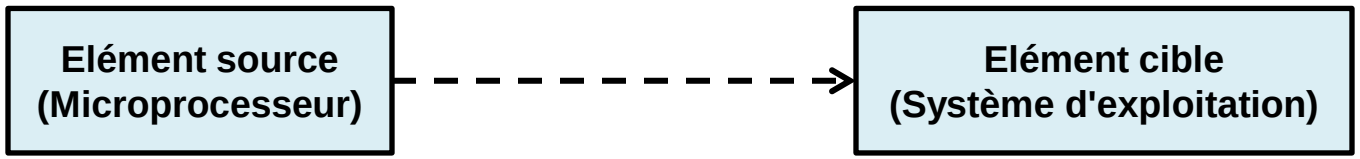
# Les contraintes

- Les **contraintes** sont des **instructions** exprimées sous forme de **langage textuel**, pouvant être **naturel** ou **formel**.
- Elles peuvent être **attachées** à n'importe quel **élément** d'un **modèle** ou à une **liste** d'éléments de **modèle**.
- Une **contrainte** est **représentée** sous la forme d'une **chaîne de texte** placée **entre** des **accolades {}**.
- Cette **chaîne de texte** correspond au **corps** de la **contrainte**, écrit dans un langage de **contrainte** qui peut être **naturel**, dédié comme le langage **OCL (Object Constraint Language)**, ou directement **issu** d'un langage de **programmation**.

L'objectif d'une contrainte est d'établir une condition ou une restriction sémantique. Elle permet de spécifier des règles ou des conditions que doivent respecter les éléments d'un modèle ou d'une liste d'éléments.

# La relation de dépendance

- La relation de **dépendance**, symbolisée par une **flèche pointillée**, établit une relation **unidirectionnelle** d'utilisation entre **deux éléments** de **modélisation**. Ces éléments sont **désignés** respectivement comme la **source** et la **cible** de la **relation**.
- La relation de **dépendance** permet **d'exprimer** que la **réalisation** ou la **mise en œuvre** d'un élément (**la cible**) **dépend** de **l'existence** ou de **l'utilisation** d'un autre élément (**la source**).




En plus de la **flèche pointillée**, une **relation de dépendance** peut être **accompagnée** d'une **note** et/ou d'une **contrainte** ou d'un **stéréotype**.

# Dichotomies (type, instance) et (type, classe)

- La **dichotomie (type, instance)** concerne la distinction entre un **type général** et une **entité spécifique** qui en est un exemple. Par exemple, le type ‘’**Animal**’’ est une catégorie générale, et ‘’**Chien**’’ est une instance spécifique de cet animal. Dans ce cas, ‘’**Animal**’’ est le type et ‘’**Chien**’’ est l'instance de ce type.

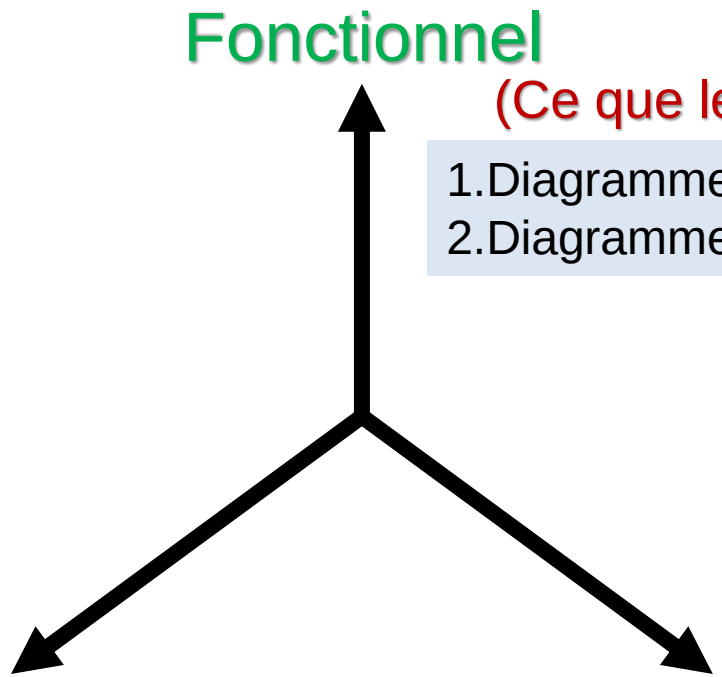
- De même, la **dichotomie (type, classe)** concerne la **distinction** entre un **type général** et une **sous-catégorie** spécifique a l'intérieur de ce type. Par exemple, le type ‘’**véhicule**’’ est une **catégorie générale**, et ‘’**Voiture**’’ est une **classe spécifique** de véhicule. Dans ce cas, ‘’**véhicule**’’ est le type et ‘’**Voiture**’’ est la **classe** a l'intérieur de ce type.

# Définitions

- **UML** comprend différents **types** de **diagrammes** qui représentent différentes **vues** pour représenter des **concepts spécifiques** d'un système d'information. Il y a **trois axes** de **modélisation** :
- L'**axe fonctionnel** se concentre sur la représentation des fonctionnalités et des comportements du système. Les diagrammes les plus couramment utilisés dans cet axe sont les diagrammes de **cas d'utilisation** et les diagrammes **d'activités**.
- L'**axe statique** se concentre sur la représentation de la structure du système, notamment les classes, les objets, les relations et les interfaces. Les diagrammes les plus couramment utilisés dans cet axe sont les diagrammes de classes, les diagrammes **d'objets**, les diagrammes de **composants** et les diagrammes de **déploiement**.
- L'**axe dynamique** se concentre sur la représentation du comportement dynamique du système. Les diagrammes les plus couramment utilisés dans cet axe sont les diagrammes de **séquence**, les diagrammes d'**états-transitions** et les diagrammes **d'interaction**.

# Les diagrammes UML

## Les 3 axes de modélisation d'un système



# Définitions (suite)

En utilisant ces **différents types** de diagrammes **UML**, les concepteurs et les développeurs peuvent **représenter** de manière **complète** et cohérente les différents **aspects** d'un **système d'information**, en fournissant des vues **fonctionnelles**, **statiques** et **dynamiques**. Cela facilite la **compréhension**, la **communication** et la **documentation** des systèmes logiciels.

# Définitions (suite)

- Ces diagrammes ont une utilité variable selon les cas et ne sont pas tous nécessairement utilisés lors d'une modélisation. Les diagrammes les plus utiles pour la **maîtrise d'ouvrage** comprennent les diagrammes de **cas d'utilisation**, de **classes**, d'**objets**, de **séquence**, d'**activités** et d'**états-transitions**.

- D'autre part, les diagrammes de **composants** et de **déploiement** sont particulièrement utiles pour la **maîtrise d'œuvre** car ils permettent de formaliser les contraintes de réalisation et la solution technique.

# Définition

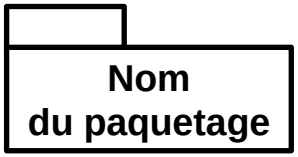
- Un **paquetage** est une unité d'organisation dans un modèle qui regroupe des éléments de modèle et des diagrammes. Il permet de structurer et de regrouper les éléments de modélisation en ensembles logiques. Un paquetage peut contenir différents types d'éléments de modèle tels que des **classes**, des **cas d'utilisation**, des **interfaces**, des **diagrammes**, etc., et il peut également contenir des **paquetages imbriqués**.

- Les **éléments** contenus dans un **paquetage** doivent être **étroitement liés** et cohérents entre eux, généralement de la même nature et du même niveau sémantique. En général, il y a une seule **racine de paquetage** qui représente l'ensemble du modèle d'un système.

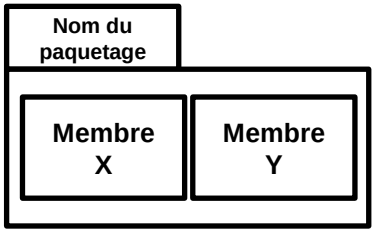
# Notation graphique

Le formalisme **général** d'un **paquetage** et les **trois** façons **distinctes** pour sa **présentation** sont les suivantes :

- **Représentation globale :** La représentation **globale** d'un **paquetage** consiste à placer le nom du paquetage à l'intérieur d'un grand rectangle. Cela permet d'identifier rapidement le paquetage et de le distinguer des autres éléments du système

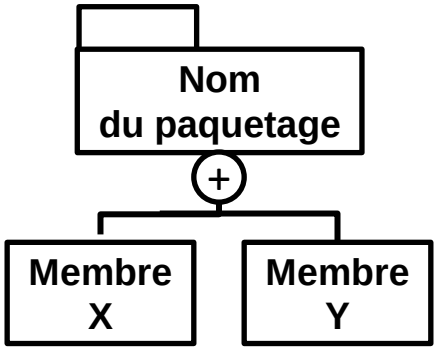



- **Représentation détaillée** : La représentation **détaillée** d'un **paquetage** comprend la visualisation des membres du paquetage, tels que les classes, les interfaces ou les sous-paquets. Le nom du paquetage global est inscrit dans un petit rectangle, ce qui permet de regrouper les membres du paquetage et de montrer leur relation.

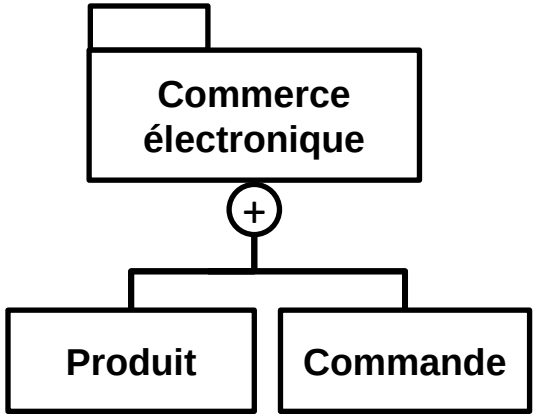


# Notation graphique

- **Représentation éclatée** : La représentation **éclatée** d'un **paquetage** consiste à afficher les membres du paquetage reliés par un lien connecté au paquetage global par le symbole ⊕. Cette représentation met l'accent sur les détails internes du paquetage et montre comment ses membres sont connectés entre eux.




**Exemple :**




Cette représentation éclatée met en évidence la séparation des responsabilités entre les deux paquets principaux, permettant une organisation claire et une compréhension des fonctionnalités spécifiques à chaque domaine.

# Fin du Chapitre 2.