

Course: UML

Ali ZAINOUL <ali.zainoul.az@gmail.com>

Crystal Clear Code
June 10, 2025



Table des matières

- 1 Introduction et annotations
- 2 Les différentes notions en UML
- 3 Tour de table des diagrammes en UML
 - Diagrammes de structure (statiques)
 - Diagrammes de comportement
 - Diagrammes d'interaction (dynamiques)

Table des matières

4 Diagrammes de structure

■ Diagramme de classes

- Principes fondamentaux de la OOP - rappels
- Concepts objets, et diagrammes de classes - lien avec la POO

■ Diagramme d'objets

■ Diagramme de composants

■ Diagramme de déploiement

■ Diagramme de paquets

■ Diagramme de structure composite

■ Diagramme de profils

Table des matières

Table des matières

**** Introduction ****

Introduction

- **L'analyse fonctionnelle:** approche structurante au service de l'équipe en charge de la création ou de l'amélioration d'un produit. L'objectif est d'identifier le besoin client, afin de déterminer les fonctions du produit, avant de rechercher les solutions techniques et technologiques à mettre en œuvre.
- l'UML est destiné à faciliter la conception des documents nécessaires au développement d'un logiciel orienté objet, comme standard de modélisation de l'architecture logicielle.

Annotations

- MOA (Maître d'Ouvrage/Chef de projet): celui qui décide ce qui doit être fait.
- MOE (Maître d'œuvre): celui qui décide comment le faire.
- AMOA (Assistant à Maîtrise d'Ouvrage): celui qui aide à le faire.
- CdC (Cahier de Charges): document formalisant un besoin, en détaillant les fonctionnalités attendues d'un système, d'un produit ou d'un service ainsi que les contraintes auxquelles il est soumis.

UML et domaines d'utilisation

- Développement/ programmation
- Génie logiciel
- Modélisation du comportement d'une BDD

**** Les notions de bases en UML ****

Les différentes notions en UML

On va illustrer ici les concepts les plus utilisés en UML:

- **Notion Orientation Objet:** principes d'analyse et de conception d'applications orientées objet: notions de programmation orientée objet (POO) et ses six principes.
- **Notion de comportement:** activité, évènement, message, méthode, état et cas d'utilisation.
- **Notion de relation:** notion d'agrégation, d'association, de composition, de dépendance, de généralisation et d'héritage.
- **Notion de diagrammes:** la section qui va suivre (RÉFÉRENCE PLUS BAS).

Les différentes notions en UML - Suite

- **Notion de Structure:** notion d'acteur, d'artéfact, attribut, classe, composant, interface, objet, package et propriété.
 - **Notion d'acteur:** En UML, un acteur est défini en tant qu'entité jouant le rôle d'un utilisateur ou par un système interagissant avec le système modélisé. Les acteurs apparaissent souvent dans les diagrammes de cas d'utilisation.



Figure: User

Les différentes notions en UML - Suite

■ Suite:

- **Notion d'artefact:** Un artefact est un élément concret du monde réel (un document, un exécutable/script/fichier, une table de BDD etc.). C'est un classeur représenté par un rectangle contenant le mot-clef « artifact » suivi du nom de l'artefact. Ceci ne nous concerne pas dans le cadre de ce cours.
- **Les notions d'attribut (= membre), classe, interface, objet et package ont été vus précédemment.**
- **Notion de composant:** un composant décrit l'organisation du système du point de vue des éléments logiciels: modules, données ou encore composants de configuration.

Les diagrammes en UML

- Diagrammes de structure (ou statiques) ;
- Diagrammes de comportement ;
- Diagrammes d'interaction (ou dynamiques).

Diagrammes de structure

- **Diagramme de classes** (class diagram): représentation des classes qui interviennent dans le système.
- **Diagramme d'objets** (object diagram): représentation des instances de classes (objets) utilisées dans le système
- **Diagramme de composants** (component diagram): représentation des composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)
- **Diagramme de déploiement** (deployment diagram): représentation des éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.

Diagrammes de structure - Suite

- **Diagramme des paquets** (package diagram) : représentation des dépendances entre les paquets (un paquet étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML), c'est-à-dire entre les ensembles de définitions.
- **Diagramme de structure composite** (composite structure diagram) : représentation sous forme de boîte blanche des relations entre composants d'une classe (depuis UML 2.x).
- **Diagramme de profils** (profile diagram) : spécialisation et personnalisation pour un domaine particulier d'un meta-modèle de référence d'UML (depuis UML 2.2).

Diagrammes de comportement

- **Diagramme des cas d'utilisation** (use-case diagram) : représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire de toutes les fonctionnalités que doit fournir le système.
- **Diagramme états-transitions** (state machine diagram) : représentation sous forme de machine à états finis du comportement du système ou de ses composants.
- **Diagramme d'activité** (activity diagram) : représentation sous forme de flux ou d'enchaînement d'activités du comportement du système ou de ses composants.

Diagrammes d'interaction

- **Diagramme de séquence** (sequence diagram) : représentation de façon séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
- **Diagramme de communication** (communication diagram) : représentation de façon simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets (depuis UML 2.x).
- **Diagramme global d'interaction** (interaction overview diagram) : représentation des enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité) (depuis UML 2.x).
- **Diagramme de temps** (timing diagram) : représentation des variations d'une donnée au cours du temps (depuis UML 2.3).

**** Les diagrammes de structure ****

**** Diagramme de classes ****

Diagramme de classes – définition

- Le diagramme de classes est un outil de modélisation UML représentant la structure statique du système.
- Il décrit les classes du système, leurs attributs, leurs méthodes ainsi que les relations entre elles (association, héritage, composition, etc.).
- Contrairement au diagramme d'objets, il ne montre pas d'instances mais plutôt une vision abstraite et globale de l'architecture.

Utilité du diagramme de classes

- Essentiel en phase de conception pour structurer le système et définir les responsabilités de chaque classe.
- Sert de base pour la génération de code, la documentation technique ou la communication entre développeurs.
- Permet d'identifier clairement les relations hiérarchiques et les dépendances logiques entre les entités du modèle.

Principes fondamentaux de la OOP - rappels

- La Programmation Orientée Objet (POO) est un des principes fondamentaux en programmation informatique, ses origines remontent aux années 1970 avec les langages Simula et Smalltalk, mais le principe a rapidement pris son envol grâce à la création du langage C++ qui est l'extension du langage C, avec en effet, cette quête de rendre les logiciels plus robustes.

Principes fondamentaux de la OOP - rappels

Une mnémotechnique utile regroupant les six concepts fondamentaux de la POO est: **ACOPIE**

- Abstraction
- Class
- Object
- Polymorphism
- Inheritance
- Encapsulation

Object

- En informatique, un objet est un conteneur (container) symbolique et autonome contenant des informations et des fonctions/méthodes concernant un sujet, manipulés dans un programme. Le sujet est souvent quelque chose de tangible appartenant au monde réel.

Class

- La classe est une structure informatique particulière dans le langage objet.
- Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type.
- Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe. C'est pourquoi les attributs d'un objet sont aussi appelés variables d'instance et les messages opérations d'instance ou encore méthodes d'instance.

Class

- L'interface de la classe (l'ensemble des opérations visibles) forme les types des objets.
- Selon le langage de programmation, une classe est soit considérée comme une structure particulière du langage, soit elle-même comme un objet (objet non-terminal). Dans le premier cas, la classe est définie dans le runtime ; dans l'autre, la classe a besoin elle aussi d'être créée et définie par une classe : ce sont les méta-classes. L'introspection des objets (ou « méta-programmation ») est définie dans ces méta-classes.
- La classe peut être décrite par des attributs et des messages. Ces derniers sont alors appelés, par opposition aux attributs et messages d'un objet, variables de classe et opérations de classe ou méthodes de classe. Parmi les langages à classes on retrouve Smalltalk, C++, C#, Java. etc.

Encapsulation

- L'encapsulation est le fait de regrouper les données et les méthodes qui les manipulent en une seule entité appelée *capsule*.
- Elle permet d'avoir un code organisé, restreindre l'accès à certaines portions depuis l'extérieur de la capsule et avoir un code robuste.

Abstraction

- L'abstraction est l'un des concepts clés dans les langages de programmation orientée objet (POO). Son objectif principal est de gérer la complexité (comprendre difficulté ici) en masquant les détails inutiles à l'utilisateur
- L'abstraction est le processus consistant à représenter un objet dans la vie réelle en tant que modèle informatique.
- Cela consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l'on souhaite manipuler, et à les placer dans un modèle informatique convenable.

Inheritance

- C'est un mécanisme pour transmettre toutes les méthodes d'une classe dite "mère" vers une autre dite "fille" et ainsi de suite.

Polymorphism

- Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

Concepts objets, et diagrammes de classes - lien avec la POO

- On a vu qu'un diagramme de classes a généralement trois compartiments:
 - Le premier concerne le nom de la classe
 - Le second concerne les membres/attributs de la classe.
 - Tandis que le troisième, concerne les méthodes/opérations de la classe.

Concepts objets, et diagrammes de classes - lien avec la POO

- La figure ci-dessous explicite la signature d'un diagramme de classes d'un point de vue global:

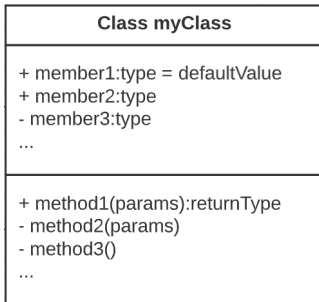


Figure: classDiag

Concepts objets, et diagrammes de classes - lien avec la POO

- La figure ci-dessous explicite les diverses notions de relation existantes:







Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Figure: relationships

**** Diagramme d'objets ****

Diagramme d'objets — définition

- Le diagramme d'objets est un instantané du système à un moment donné, représentant des instances concrètes de classes (appelées objets).
- Il permet de visualiser l'état des objets et leurs liens à un moment spécifique de l'exécution, contrairement au diagramme de classes qui décrit des structures générales.

Utilité du diagramme d'objets

- Très utile en phase d'analyse pour expliquer des cas concrets, des scénarios de test, ou représenter un état du système réel.
- Idéal pour valider la conception des classes et leurs relations, en observant les objets instanciés et leurs valeurs.

**** Diagramme de composants ****

Diagramme de composants – définition

- Le diagramme de composants est utilisé pour modéliser la structure physique d'un système logiciel.
- Il décrit les composants logiciels (fichiers exécutables, bibliothèques, modules, etc.) ainsi que leurs interfaces et dépendances.

Utilité du diagramme de composants

- Recommandé dans les phases d'architecture logicielle et d'intégration, notamment pour les projets modulaires ou orientés microservices.
- Il distingue les composants fournisseurs (provides) et requérants (requires), facilitant la compréhension de l'encapsulation logicielle.

**** Diagramme de déploiement ****

Diagramme de déploiement — définition

- Ce diagramme modélise l'infrastructure physique qui supporte un système logiciel.
- Il décrit les nœuds matériels (serveurs, ordinateurs, appareils embarqués) ainsi que les artefacts déployés (logiciels, composants).

Utilité du diagramme de déploiement

- Utile pour aborder la répartition physique du système, notamment dans les architectures distribuées (ex. : cloud, client/serveur, IoT).
- Permet de comprendre les contraintes d'exécution, comme la bande passante, la sécurité ou la scalabilité.

**** Diagramme de paquets ****

Diagramme des paquets — définition

- Le diagramme de paquets permet d'organiser les classes ou éléments UML dans des groupes logiques appelés « paquets ».
- Il reflète la structure logique et hiérarchique du projet.

Utilité du diagramme de paquets

- Précieux pour modulariser le code, clarifier l'architecture logique, et maîtriser les dépendances entre modules.
- En programmation Java, C#, Python ou C++, le paquet (package / namespace) est une unité de structuration du code source.

**** Diagramme de structure composite ****

Diagramme de structure composite – définition

- Ce diagramme permet de détailler l'organisation interne d'une classe ou d'un composant.
- Il met en évidence les parties internes, les ports de communication, et les liens d'interaction.

Utilité du diagramme de structure composite

- Excellent pour illustrer l'architecture interne de composants complexes ou de systèmes embarqués.
- Souvent utilisé avec des architectures orientées composants (ex. : IoC, Spring, OSGi) pour montrer les flux internes de communication.

**** Diagramme de profils ****

Diagramme de profils – définition

- Ce diagramme permet d'étendre UML en définissant des stéréotypes, des contraintes et des valeurs par défaut à travers des profils UML.
- Il est indispensable pour créer des domaines spécifiques (DSML – Domain-Specific Modeling Languages).

Utilité du diagramme de profils

- Outil avancé pour les experts souhaitant adapter UML à des cas métiers spécifiques (par exemple, dans l'aérospatial, la médecine ou l'automobile).
- Un profil UML permet de définir un vocabulaire personnalisé, tout en restant conforme à la méta-modélisation UML.