



M201

Préparer un projet web

SOMMAIRE



1. Modélisation d'un projet web
2. Représentation de la vue dynamique d'un système
3. Création des maquettes pour le développement web
4. Préparation de l'environnement de développement web



PARTIE 1

Modélisation d'un projet web

Dans cette partie, vous allez :

- ✓ Appréhender le cycle de vie d'un projet web
- ✓ Modéliser les besoins client par un diagramme de cas d'utilisation
- ✓ Modéliser les données du projet par un diagramme de classes
- ✓ Modéliser les interactions d'objets d'un système à l'aide d'un diagramme de séquence
- ✓ Elaborer des diagrammes UML à l'aide d'un outil de modélisation



CHAPITRE 1

Appréhender le cycle de vie d'un projet web

Ce que vous allez apprendre dans ce chapitre :

- ✓ Cycle de vie d'un projet (Définition, étapes,...)
- ✓ Analyse des besoins
- ✓ Conception
- ✓ Développement
- ✓ Tests et déploiement
- ✓ Maintenance et évolutivité

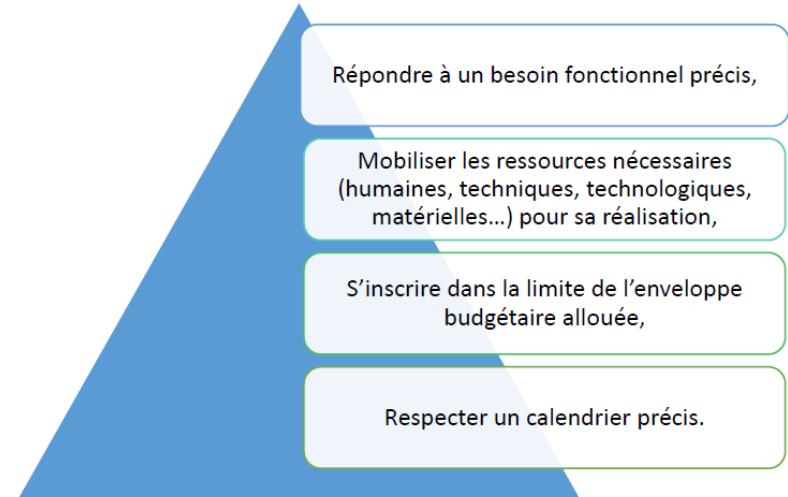
- **La conception d'applications web** consiste à planifier et structurer des solutions numériques pour répondre à des besoins spécifiques.
- **La modélisation** permet de représenter visuellement ces solutions à l'aide d'outils comme **UML (Unified Modeling Language)**, qui standardise les diagrammes pour mieux comprendre et communiquer les composants d'un projet.



Un projet web est une initiative visant à développer des applications ou des sites web pour répondre à des besoins spécifiques, que ce soit pour informer, vendre, communiquer ou fournir des services en ligne.

Ses objectifs peuvent inclure la création de services en ligne, l'amélioration de l'expérience utilisateur, ou l'automatisation de processus.

Le projet web doit :



Objectifs d'un projet web

Avant de démarrer un projet web, il est impératif pour l'entreprise de définir clairement ce que l'on attend concrètement de son projet web. Voici quelques objectifs fondamentaux

La notoriété, la crédibilité, l'e-réputation

La visibilité, la popularité

La mobilité, le Responsive Web Design

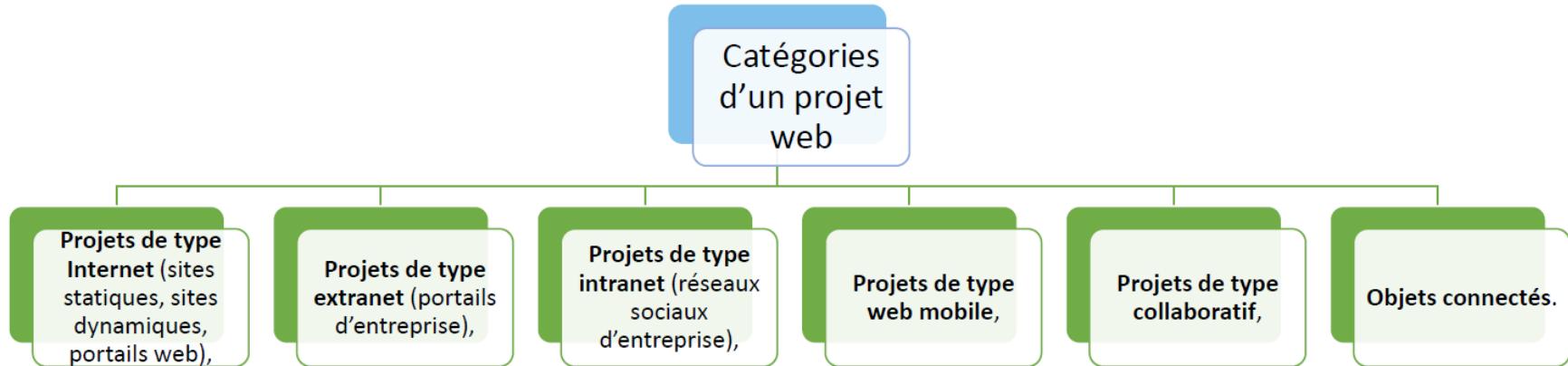
La qualité et la pérennité

La rentabilité et la productivité

La simplicité,

L'utilisabilité.

Catégories d'un projet web



Catégories d'un projet web :

Projets de type Internet (sites statiques, sites dynamiques, portails web)

Sites Statiques : Ce sont des sites où le contenu ne change pas souvent. Les pages sont fixes et affichent des informations qui ne nécessitent pas d'interaction ou de mise à jour en temps réel.

Sites Dynamiques : Ces sites peuvent afficher des informations qui changent régulièrement. Ils utilisent des bases de données pour personnaliser le contenu selon les besoins des utilisateurs.

Portails Web : Ce sont des sites qui regroupent des informations provenant de différentes sources. Ils peuvent inclure des actualités, des forums, des outils de recherche, etc., souvent avec des fonctionnalités d'accès personnalisé.

Catégories d'un projet web

Projets de type Extranet :

Portails d'Entreprise : Ces portails sont réservés aux partenaires externes de l'entreprise, comme les fournisseurs ou les clients. Ils permettent de partager des informations et des documents de manière sécurisée avec ces partenaires.

Projets de type Intranet :

Réseaux Sociaux d'Entreprise : Il s'agit de plateformes internes où les employés peuvent communiquer et collaborer. Cela inclut des outils de messagerie, et des fonctionnalités de partage de documents, souvent dans un environnement sécurisé et fermé à l'extérieur.

Catégories d'un projet web

Projets de type Web Mobile :

Sites ou Applications Mobile : Ces projets sont conçus spécifiquement pour les téléphones et les tablettes. Ils peuvent être des sites web optimisés pour les mobiles ou des applications web accessibles via un navigateur mobile.

Projets de type Collaboratif :

Outils de Collaboration : Ce sont des plateformes qui permettent à plusieurs personnes de travailler ensemble sur des projets en ligne. Cela inclut des outils pour le partage de fichiers, la gestion de tâches, et la communication en temps réel.

Catégories d'un projet web

Objets Connectés :

Internet des Objets (IoT) : Ce sont des dispositifs physiques connectés à Internet, comme des montres intelligentes ou des capteurs de maison. Ils envoient des données et reçoivent des commandes via des interfaces web.



01 - Appréhender le cycle de vie d'un projet web

1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Définition du Cycle de Vie d'un Projet Web

Le cycle de vie d'un projet web est l'ensemble des phases que traverse un projet, depuis sa **conception** initiale jusqu'à sa **réalisation** finale et son **maintien**. Chaque phase joue un rôle spécifique dans la réussite du projet.

Il contient généralement 4 phases (Cadrage, Conception, Réalisation et Clôture).

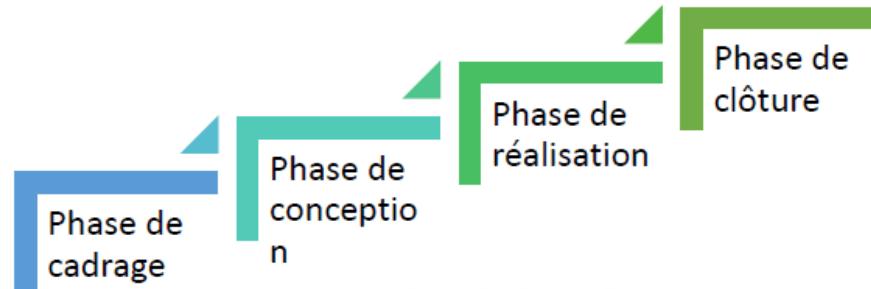


Figure 1 : Les 4 étapes du cycle de vie d'un projet

Phase de cadrage : elle a pour objectif de cadrer le projet, identifier le besoin à l'origine du projet, ses enjeux et ses objectifs, le contexte du projet, le périmètre et ses limites.

Phase de conception : nommée aussi phase de planification ou phase de préparation, lors de cette étape, le chef de projet, accompagné de l'équipe projet, définit qui fait quoi, quand et comment.

Phase de réalisation : C'est lors de cette phase que les actions sont réalisées, dans le respect du planning, du budget et des échéances fixées avec le client ou le commanditaire du projet.

Phase de clôture : phase de finalisation et conclure du projet, en s'assurant que tous les objectifs ont été atteints et que le projet est livré avec succès.

Modèles du cycle de vie

Il existe plusieurs modèles du cycle de vie de projet, chacun ayant ses propres caractéristiques et processus adaptés à différents types de projets.

Modèle en Cascade :

Le modèle en cascade est linéaire et séquentiel. Les étapes du projet sont réalisées de manière successive, chaque phase devant être terminée avant de passer à la suivante.

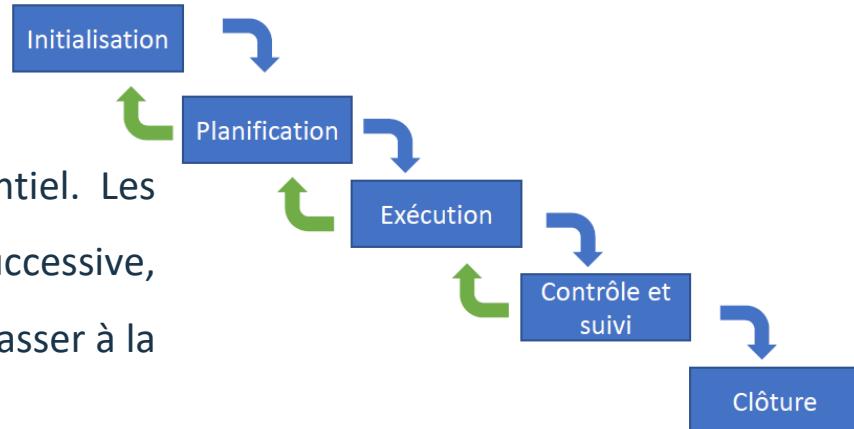


Figure 2 : Cycle de vie en cascade

Modèles du cycle de vie

Modèle Agile :

Le modèle Agile est itératif et flexible. Les projets sont divisés en petites unités appelées "sprints" ou "itérations", permettant des ajustements réguliers et des améliorations en cours de route.



Modèle en V :

Le modèle en V est similaire au modèle en cascade mais avec un accent mis sur les tests à chaque étape du développement. Chaque phase de développement a une phase de test correspondante.

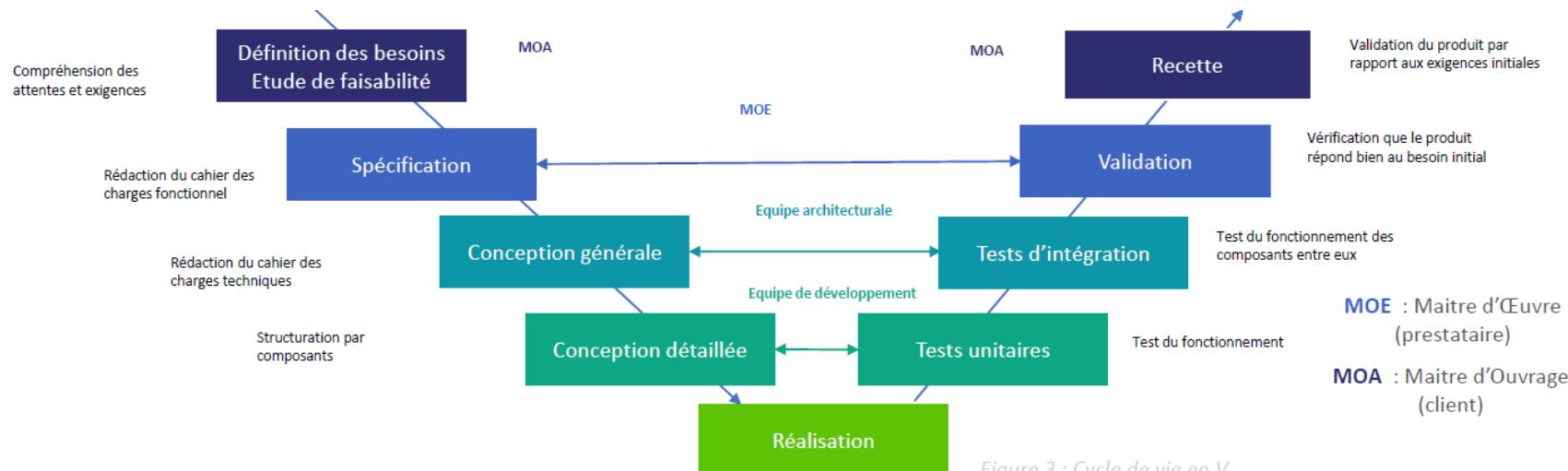


Figure 3 : Cycle de vie en V



Figure 4 : Cycle de vie d'un projet web



01 - Appréhender le cycle de vie d'un projet web

1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Définition des objectifs et besoins dans un cahier de charges

Avant de se lancer dans la conception et la réalisation d'un projet web, il est recommandé d'établir clairement les objectifs, la liste des besoins et les résultats attendus dans un cahier de charge pour assurer la réussite du projet.

Un **cahier des charges** est un document formel qui décrit les **spécifications du projet**, les **attentes** des parties prenantes, les **fonctionnalités** requises, et les **contraintes techniques** et **organisationnelles**. Il sert de référence pour toutes les phases du projet, du développement à la mise en production.

Eléments clés d'un cahier de charges

- Le cahier des charges est probablement le document le plus important dans un projet web.
- Il est rédigé par la maîtrise d'ouvrage (MOA) dans un langage accessible et non technique, afin de communiquer clairement les attentes du projet à la maîtrise d'œuvre (MOE).
- La structure et le contenu du cahier des charges dépendent du type de projet web et des spécificités de l'entreprise (MOA). il doit contenir :

- 1 Présentation de l'entreprise et de son organisation
- 2 Contexte
- 3 Objectifs du projet web
- 4 Eléments fonctionnels et techniques
- 5 Planning, budget, qualité et organisation du projet
- 6 Eléments juridiques



L'analyse des besoins, également appelée spécifications des besoins, est une étape essentielle dans la préparation d'un projet web. Elle vise à comprendre et définir précisément ce que le projet doit accomplir pour répondre aux attentes des utilisateurs et des parties prenantes. Cette analyse se décompose généralement en trois interventions complémentaires :



Analyse des Besoins Métiers :

Objectifs : Identifier et comprendre les objectifs stratégiques et opérationnels de l'entreprise. Cette analyse se concentre sur ce que l'entreprise veut atteindre avec le projet web (par exemple, améliorer la visibilité en ligne, augmenter les ventes, ou fournir un service particulier...).

Résultat : Un ensemble de besoins et d'exigences métier qui guideront le développement du projet.

Analyse des Besoins Utilisateurs :

Objectifs : Se mettre à la place des utilisateurs pour comprendre leurs attentes, leurs comportements et leurs besoins spécifiques lorsqu'ils interagissent avec le site web.

Méthodes : Utiliser des scénarios d'utilisation, des enquêtes, et des tests utilisateur pour capturer les besoins réels des utilisateurs finaux.

Résultat : Des spécifications qui reflètent ce que les utilisateurs attendent du site, comme une navigation intuitive, des temps de chargement rapides, ou des fonctionnalités spécifiques.

Analyse Fonctionnelle :

Objectif : Définir les fonctionnalités spécifiques que le site doit offrir pour répondre aux besoins métiers et utilisateurs identifiés.

Contenu : Décrire les fonctionnalités requises (par exemple, système de gestion de contenu, formulaires de contact, processus d'achat en ligne) et les spécifications techniques associées.

Résultat : Un document détaillé qui sert de guide pour les développeurs, indiquant comment chaque besoin doit être implémenté dans le site web.



01 - Appréhender le cycle de vie d'un projet web

1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

La conception vise à planifier en détail le développement du projet. Cela inclut la définition des fonctionnalités, l'organisation des ressources, la planification des tâches, et la création de prototypes ou maquettes pour visualiser le produit final.

Cette phase permet de s'assurer que tous les aspects du projet sont bien alignés avec les besoins identifiés et les objectifs fixés.

La conception implique les facettes suivantes:



Conception fonctionnelle



Conception graphique



Conception technique



Conception fonctionnelle

La **conception fonctionnelle** décrit **comment** le produit doit fonctionner en termes de fonctionnalités et d'interactions, sans se concentrer sur les aspects techniques ou le design graphique. C'est comme créer un plan détaillé de ce que le produit doit faire avant de passer à la construction réelle.



Conception graphique

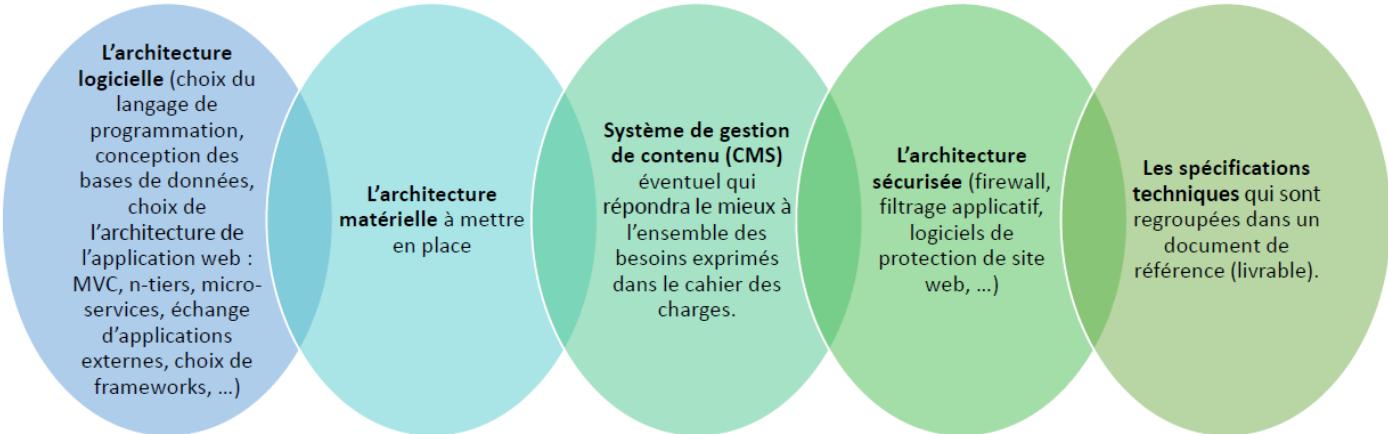
La **conception graphique** est le processus de création de l'apparence visuelle d'un site web ou d'une application. Cela inclut les couleurs, les polices de caractères, les images, les icônes, et la mise en page générale.

L'objectif est de créer une interface attrayante et cohérente qui améliore l'expérience utilisateur et reflète l'identité de la marque.



Conception technique

La **conception technique** est le processus de planification et de définition des aspects techniques nécessaires pour construire le site web ou l'application. Cela inclut la sélection des technologies, la définition de l'architecture du système, et la création des spécifications détaillées pour les développeurs.





01 - Appréhender le cycle de vie d'un projet web

1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

La **phase de développement** est le processus de création du site web ou de l'application à partir des conceptions et des spécifications élaborées. C'est là que le code est écrit, les fonctionnalités sont intégrées, et les composants sont assemblés pour réaliser le produit final.

Elle est composée des étapes suivantes :

Réalisation technique

Réalisation graphique

Réalisation éditoriale

Phase I : Réalisation technique

Elle comprend :

- **L'infrastructure de réalisation** (Mise en place de l'environnement de développement et frameworks, environnement serveurs et réseau)
- **Mise en place et alimentation des bases de données.** En cas de données telles que photos, publications sur réseaux sociaux, on se base sur le cloud.
- **Développement et codage**



- **Intégration de l'existant** : évolution d'un site web existant (reprise, amélioration, migration)

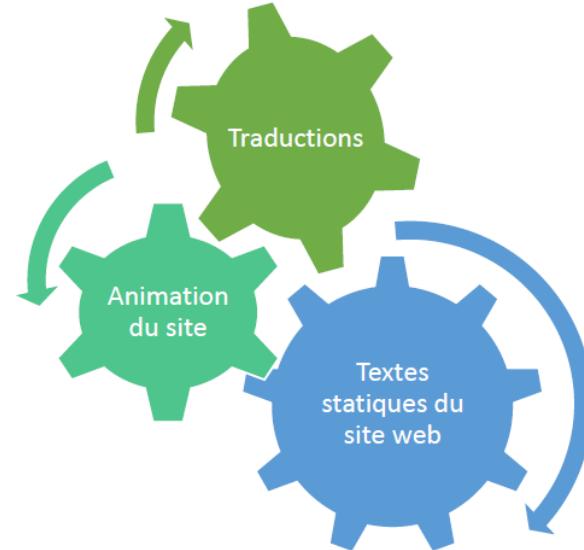
Phase II : Réalisation graphique

Elle comprend :

- Les éléments visuels (contenu visuel) :
 - Sélection d'images avec pertinence, simplicité et uniformité
 - Positionnement des éléments : bien placer les éléments et parvenir à une composition équilibrée
 - Choix d'une bonne palette de couleurs
 - Représentation visuelle des données statistiques par exemple
- Les bibliothèques d'images pour illustrer les contenus web, avec des résolutions adaptées à une navigation via réseau, mobile
- Bien choisir ses visuels : susciter l'action, mélanger texte et image, optimiser la taille des images
- Optimiser les visuels pour le référencement naturel, et ainsi une meilleure visibilité du contenu dans les moteurs de recherche (enrichissement textuel des contenus visuels : attribut, alt, titre, description, ...)

Phase III : Réalisation éditoriale

La phase de réalisation éditoriale est une étape clé dans le cycle de vie d'un projet web, surtout lorsqu'il s'agit de sites web ou d'applications qui nécessitent du contenu écrit. Cette phase se concentre sur la création, la gestion, et la publication de contenu pertinent et attrayant pour le site ou l'application.





01 - Appréhender le cycle de vie d'un projet web

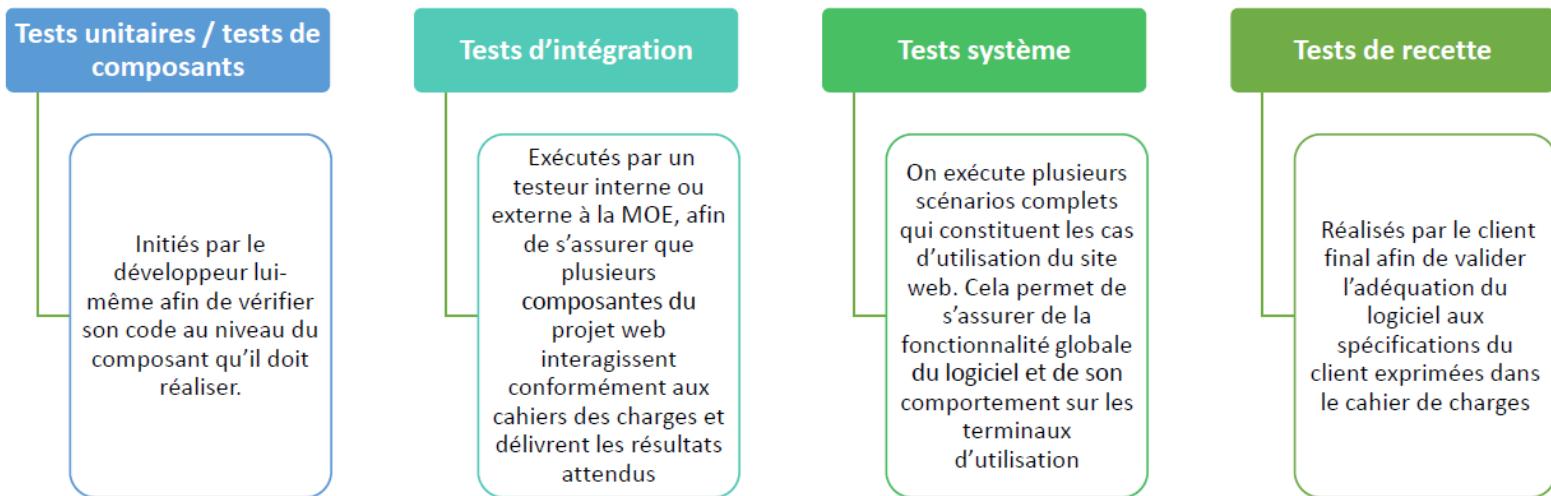
1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Tests

La phase de test du projet web favorise la détection de tous les bugs pour assurer la qualité de l'application. Il permet de rationaliser les coûts de développement du site web grâce à la maîtrise et à la correction en amont des défauts fonctionnels.

Elle garantit également l'acceptabilité du projet à la livraison lors de phase de recette chez le MOA.

Il existe 4 niveaux de tests :



Tests

Les tests peuvent être classés également en **famille de tests suivant leur nature**. Les deux familles principales :

- **Tests fonctionnels :**

Tests fonctionnels

On teste le comportement du site vis-à-vis des fonctionnalités souhaitées et attendues par le client

Tests unitaires

Tests d'intégration

Tests de système

Tests de recette

- **Tests non fonctionnels.** On peut citer :

Tests non fonctionnels

Tests de robustesse
tester le comportement du site web dans des cas "extrêmes" (forte activité, disponibilité ...)

Tests de performance
tester par exemple : consommation CPU, exploitation de mémoire RAM, volume de commandes lancées à la seconde ou encore mouvement d'entrée et de sortie des utilisateurs sur le site

Tests de montée en charge
tester sa capacité à supporter de plus en plus d'internautes tout en maintenant une expérience utilisateurs optimale et un fonctionnement correspondant aux cahier des charges

Tests de compatibilité de plateforme
vérifier le bon fonctionnement du logiciel sur des terminaux cibles notamment les systèmes d'exploitation après leur installation, les navigateurs clients

Tests d'ergonomie
évaluer l'expérience utilisateur (UX) côté design, esthétique, visuel, etc.

Tests d'interface graphique
s'assurer que la présentation graphique est suffisamment attrayante pour être accepté

Tests de sécurité
définir les niveaux de sécurité et prévoir les tests de sécurité associés

Tests

Les tests peuvent être classés également en **famille de tests suivant leur nature**. Les deux familles principales :

- **Tests non fonctionnels.** On peut citer :

Tests non fonctionnels

Tests de robustesse tester le comportement du site web dans des cas "extrêmes" (forte activité, disponibilité ...)	Tests de performance tester par exemple : consommation CPU, exploitation de mémoire RAM, volume de commandes lancées à la seconde ou encore mouvement d'entrée et de sortie des utilisateurs sur le site	Tests de montée en charge tester sa capacité à supporter de plus en plus d'internautes tout en maintenant une expérience utilisateurs optimale et un fonctionnement correspondant aux cahier des charges	Tests de compatibilité de plateforme vérifier le bon fonctionnement du logiciel sur des terminaux ciblés notamment les systèmes d'exploitation après leur installation, les navigateurs clients	Tests d'ergonomie évaluer l'expérience utilisateur (UX) côté design, esthétique, visuel, etc.	Tests d'interface graphique s'assurer que la présentation graphique est suffisamment attrayante pour être accepté	Tests de sécurité définir les niveaux de sécurité et prévoir les tests de sécurité associés
--	--	--	---	---	---	---

Déploiement

- Déployer une application ou un site web signifie **appliquer un procédé permettant d'installer ou mettre à jour le site web** sur un environnement donné.
- **Le processus de déploiement :**



- La mise en ligne est l'aboutissement du projet web. Tout ce qui a été pensé et réalisé va se retrouver en ligne, soumis à l'appréciation d'une population internaute que l'on a ciblée et pour laquelle on a essayé d'anticiper les attentes
- Actuellement, il existe de nombreux outils pour automatiser le déploiement des sites web



01 - Appréhender le cycle de vie d'un projet web

1. Cycle de vie d'un projet (Définition, étapes,...)
2. Analyse des besoins
3. Conception
4. Développement
5. Tests et déploiement
6. Maintenance et évolutivité

Les maintenances

- Un site internet a besoin d'un **entretien régulier** pour garantir tout son potentiel technique. Le web étant un environnement technologique où tout évolue très rapidement, il est d'autant plus important de faire de la maintenance **un processus continu** pour bénéficier d'un outil performant et génératrice de trafic après plusieurs années.
- Un site mal entretenu et contenant beaucoup d'erreurs projette une image peu soignée de l'entreprise, avec un impact négatif sur le référencement naturel : baisse du nombre de pages indexées par les moteurs de recherche, chute des positions dans les pages de résultats de recherche...
- La maintenance d'un site internet repose de manière générale sur deux types d'intervention :



Maintenance technique du site

- Corrections des bugs,
- Optimisation du code,
- Mises à jour des plugins, ...

Maintenance du contenu

- Ajouts de nouveaux articles d'actualités ou de blog,
- Optimisation sémantique d'une page en vue d'améliorer son référencement naturel,
- Ajouts d'images ou de contenus multimédia, ...

Maintenance technique

- Appelée également maintenance **corrective** et **évolutive**, est applicable aux sites internet après leur livraison, et formalisée sous forme de contrat entre le propriétaire du site et le prestataire (le plus souvent une agence web).
- Ces contrats de maintenance permettent de bénéficier des actions suivantes :



Vérification et **mise à jour des technologies employées** pour le fonctionnement du site internet

- Mise à jour de plugins,
- Mise à jour de la base de données...



Détection et corrections des erreurs (bugs)

- Liées à la mise en forme et à l'affiche du site web

Maintenance du contenu du site

Tout comme la maintenance technique, la mise à jour des contenus du site web est primordiale si on veut rester visible auprès des internautes/clients.

L'actualisation des contenus est en effet importante pour optimiser le référencement naturel du site web. Plus on met à jour les pages du site et on ajoute des nouveaux articles, et plus les moteurs de recherche auront tendance à valoriser le site dans les pages des résultats de recherche.

CHAPITRE 1

Modéliser les besoins client par Un diagramme de cas d'utilisation

Ce que vous allez apprendre dans ce chapitre :

- ✓ Cycle de vie d'un projet (Définition, étapes,...)
- ✓ Introduction au langage de modélisation UML
- ✓ Définition du diagramme des cas d'utilisation
- ✓ Acteurs
- ✓ Cas d'utilisation
- ✓ Relation entre acteurs et cas d'utilisation
- ✓ Relations entre cas d'utilisation
- ✓ Relation de généralisation ou de spécialisation
- ✓ Description textuelle des cas d'utilisation



02 - Modéliser les besoins client par un diagramme de cas d'utilisation



1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Modélisation

La modélisation est le processus de création d'une **représentation simplifiée et abstraite** d'un système.

Cette représentation, ou modèle, sert à mieux comprendre, analyser, et communiquer les aspects essentiels du système.

Exemple Pratique de Modélisation :

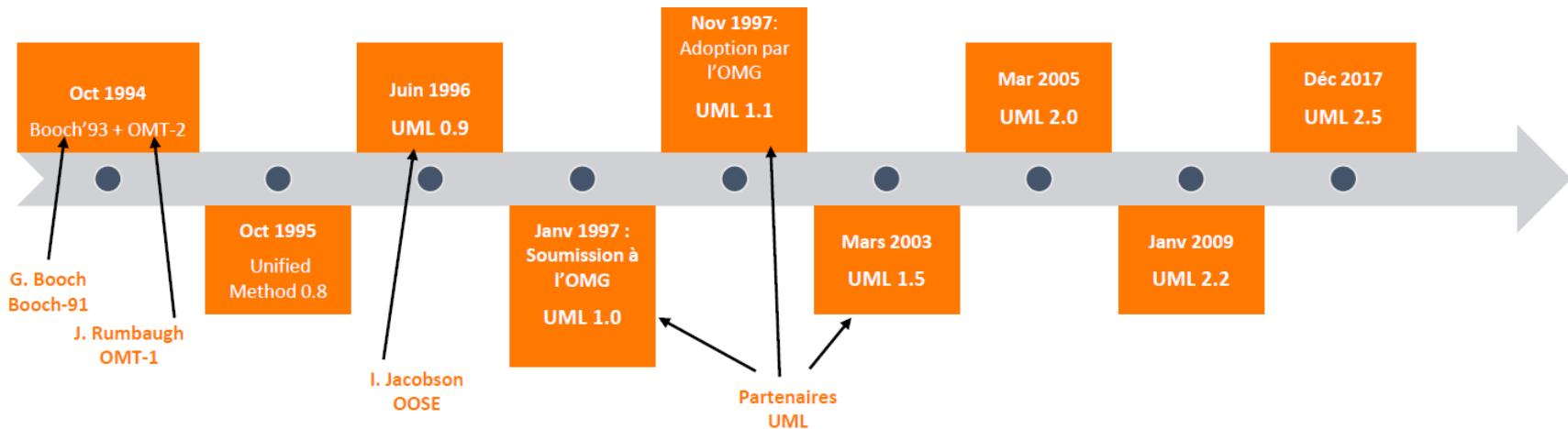
Imaginez que vous souhaitez développer une application de gestion de tâches. Avant de coder, vous commencez par modéliser l'application. Vous dessinez un diagramme qui montre les utilisateurs, les actions qu'ils peuvent effectuer (comme ajouter une tâche, supprimer une tâche), et comment ces actions interagissent avec l'application. Ce modèle vous aide à visualiser le flux de travail et à identifier les besoins fonctionnels dès le départ.

Modélisation avec UML

UML (Unified Modeling Language) est un langage de modélisation standardisé utilisé pour représenter graphiquement les systèmes logiciels. Il permet de créer des diagrammes pour décrire différents aspects d'un système, comme sa structure, son comportement, et ses interactions.

UML a été développé dans les années 1990 par Object Management Group (OMG) pour unifier plusieurs méthodes de modélisation existantes.

Historique de l'UML



UML comporte quatorze types de diagrammes, qui sont répartis en trois grandes catégories selon les aspects du système qu'ils représentent :

Niveau Fonctionnel	Niveau Structurel	Niveau comportemental
<ul style="list-style-type: none">Diagramme Use Case (Cas d'utilisation)	<ul style="list-style-type: none">Diagramme de classesDiagramme d'objetsDiagramme de composantsDiagramme de déploiementDiagramme de paquetagesDiagramme de structures composites	<ul style="list-style-type: none">Diagramme d'activitésDiagramme d'états-transitionsDiagrammes d'interactionDiagramme de séquenceDiagramme de communicationDiagramme global d'interactionDiagramme de temps

02 - Modéliser les besoins client par un diagramme de cas d'utilisation



1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Le **diagramme des cas d'utilisation (Use Case Diagram)** constitue la première étape de l'analyse UML en :

- Modélisant les besoins des utilisateurs.
- Identifiant les grandes fonctionnalités et les limites du système.
- Représentant les interactions entre le système et ses utilisateurs.

Il répond à la question : A qui et pour quoi faire?

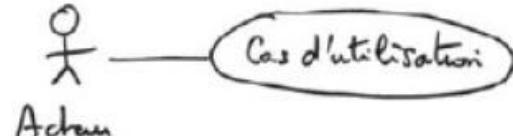
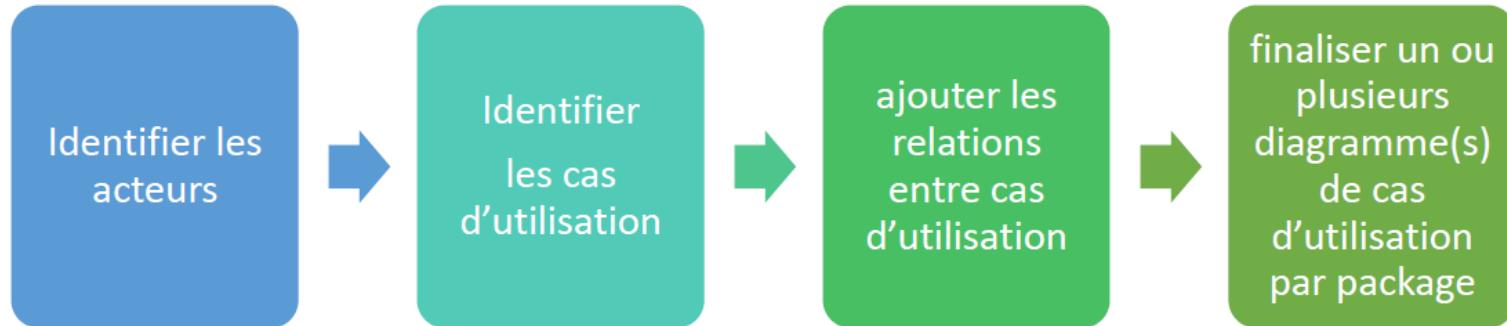


Figure 11 : Diagramme Cas d'utilisation

Diagramme des cas d'utilisation – Démarche

Comment créer un Diagramme de Cas d'Utilisation



02 - Modéliser les besoins client par un diagramme de cas d'utilisation



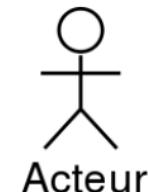
1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Acteurs

Les acteurs sont des entités externes qui interagissent avec le système. Ils peuvent être des utilisateurs humains, d'autres systèmes, ou des matériels (capteurs, moteurs...).

Les **acteurs** se représentent sous la forme d'un petit personnage (**stick man**) ou sous la forme d'une case rectangulaire (appelé **classeur**) avec le mot clé « **actor** ». Chaque acteur porte un nom.

Exemples : Client, Administrateur, Système de Paiement.



02 - Modéliser les besoins client par un diagramme de cas d'utilisation



1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Cas d'utilisation

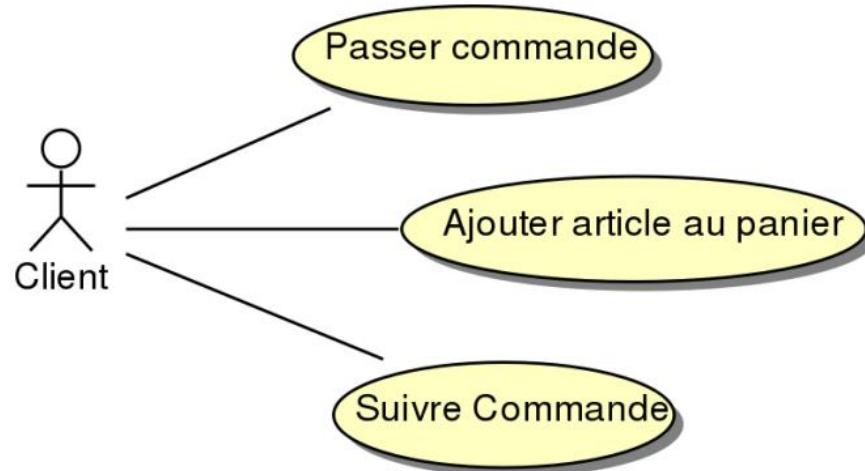
Un **cas d'utilisation** représente une fonctionnalité ou un service offert par le système en réponse à une action de l'acteur.

Un cas d'utilisation se représente par une ellipse contenant le nom du cas d'utilisation (**phrase commençant par un verbe à l'infinitif**) .

Exemples : Passer une commande, Se connecter, Effectuer un paiement.



Diagramme des Cas d'utilisation – Exemple 1



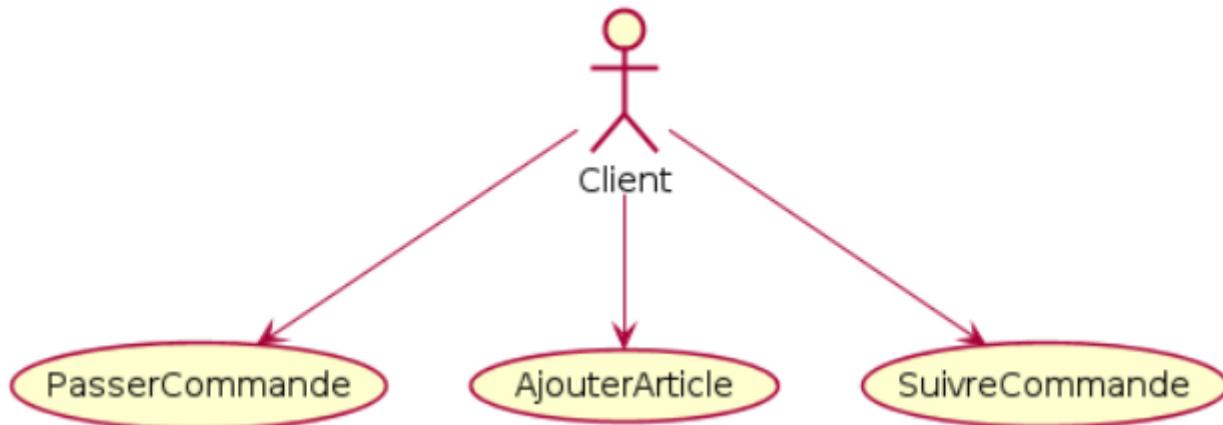
02 - Modéliser les besoins client par un diagramme de cas d'utilisation



1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Relation d'association

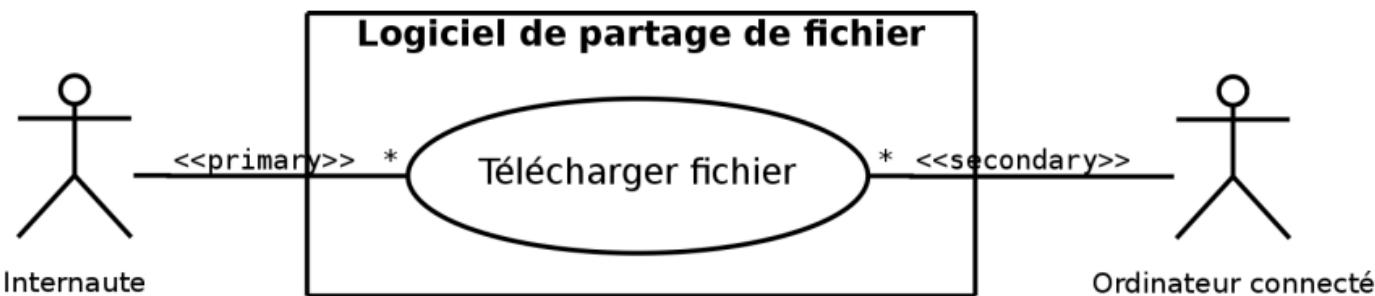
- Représente une interaction entre un acteur et un cas d'utilisation.
- Une association peut être représentée sans flèche ou avec une flèche indiquant le sens de l'interaction (vers le système ou depuis le système).



Multiplicité

Lorsqu'un acteur peut interagir plusieurs fois avec un cas d'utilisation, il est possible d'ajouter une multiplicité sur l'association du côté du cas d'utilisation.

- Le symbole ***** signifie plusieurs
- Exactement **n** s'écrit tout simplement **n**
- **n..m** signifie entre **n** et **m**.



02 - Modéliser les besoins client par un diagramme de cas d'utilisation

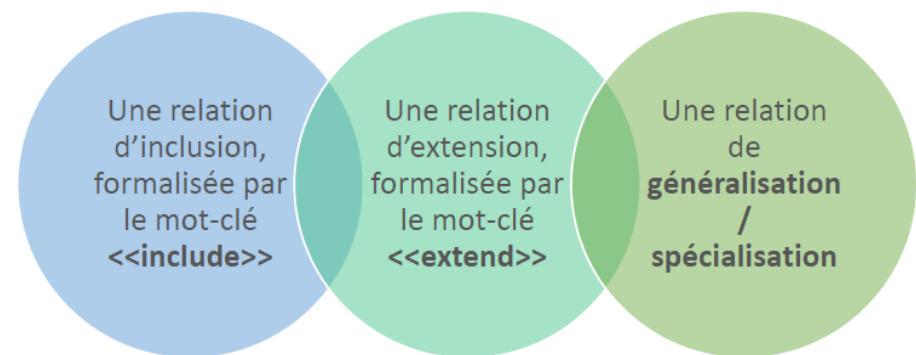


1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Relations entre cas d'utilisation

Les relations entre les cas d'utilisation montrent comment ces derniers interagissent ou dépendent les uns des autres. Ces relations permettent de comprendre la manière dont les différentes fonctionnalités du système sont connectées et comment elles peuvent être réutilisées.

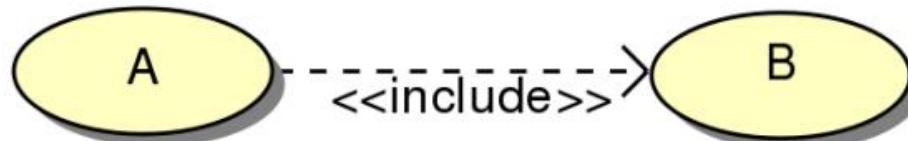
UML définit trois types de relations standardisées entre cas d'utilisation.



Relation d'inclusion ----- «include» ----- >

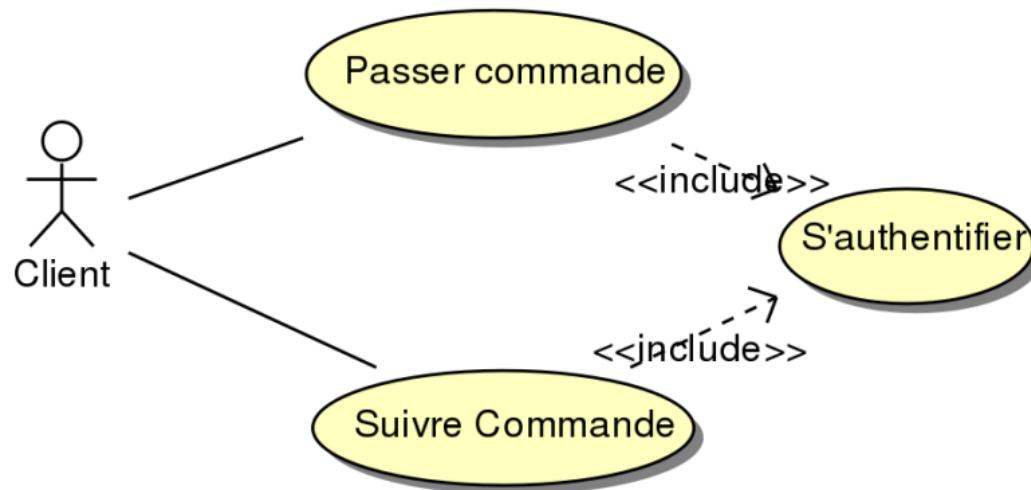
La relation « **include** » indique qu'un cas d'utilisation (appelé cas d'utilisation de base) inclut systématiquement un autre cas d'utilisation (appelé cas d'utilisation inclus) pour accomplir une partie de ses fonctionnalités. Cette relation est utilisée pour éviter la duplication des comportements communs dans plusieurs cas d'utilisation.

cette relation est représentée par une flèche pointillée reliant les 2 cas d'utilisation et munie du stéréotype « **include** ».



le cas A inclut le cas B (B est une partie obligatoire de A).

Relation d'inclusion - Exemple



Relation d'extension ----- «extend» ----- >

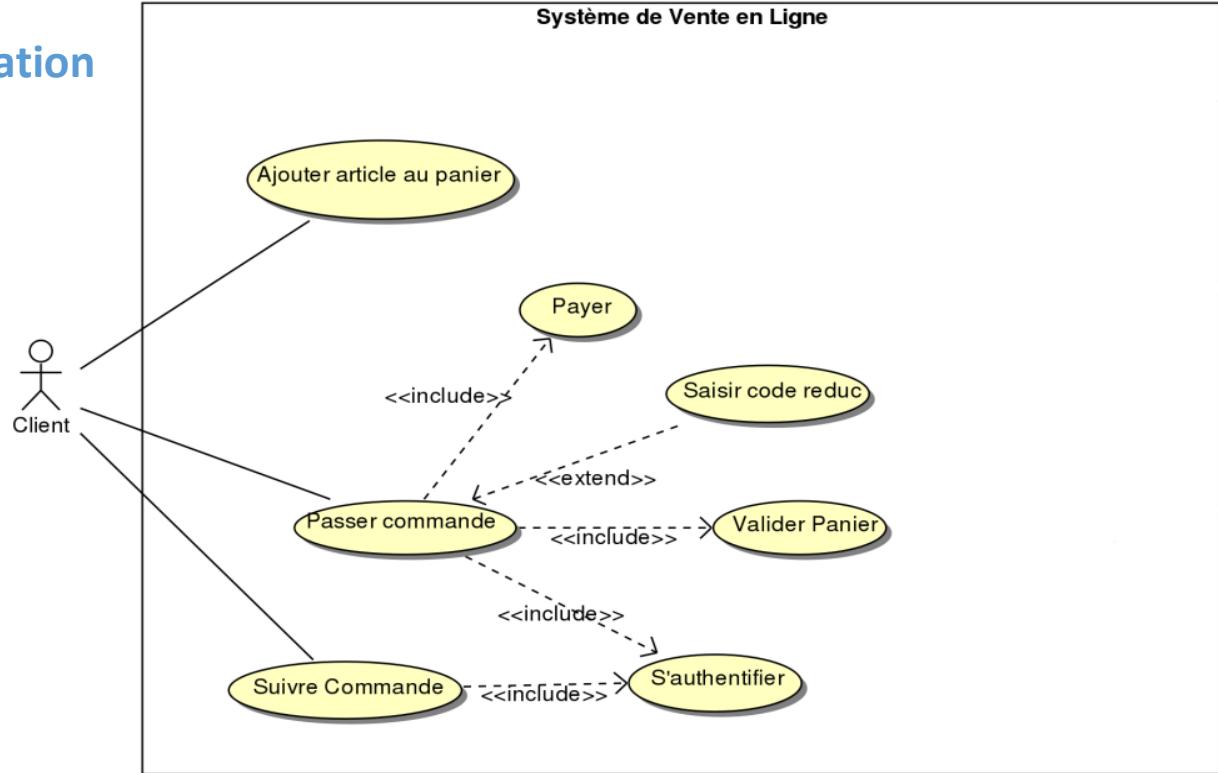
La relation « **extend** » indique que le comportement d'un cas d'utilisation de base peut être étendu ou complété par un autre cas d'utilisation (appelé cas d'utilisation étendu) de manière conditionnelle. Cette relation est utilisée pour ajouter des fonctionnalités optionnelles ou spécifiques selon certaines conditions.

Cette relation est représentée par une flèche en pointillée reliant les 2 cas d'utilisation et munie du stéréotype « **extend** ».



le cas B étend le cas A (B est une partie optionnelle de A).

Diagramme des Cas d'utilisation



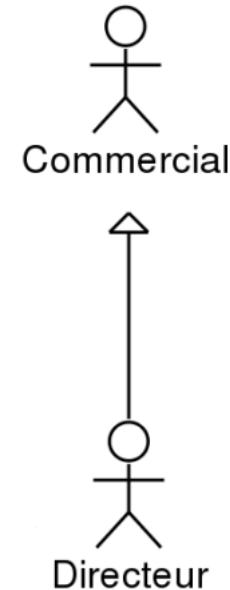
02 - Modéliser les besoins client par un diagramme de cas d'utilisation



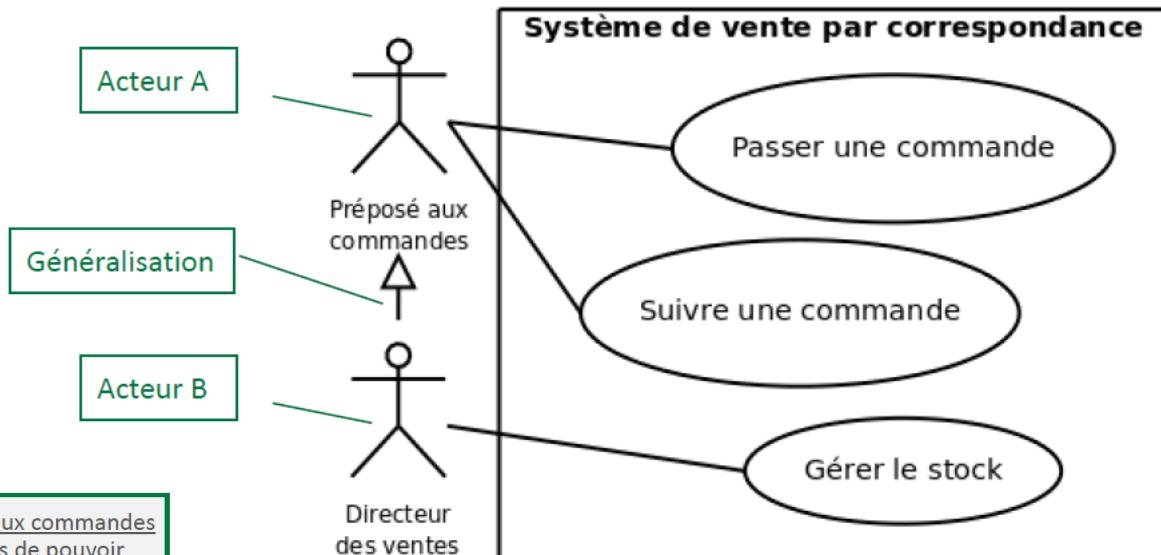
1. Introduction au langage de modélisation UML
2. Définition du diagramme des cas d'utilisation
3. Acteurs
4. Cas d'utilisation
5. Relation entre acteurs et cas d'utilisation
6. Relations entre cas d'utilisation
7. Relation de généralisation ou de spécialisation
8. Description textuelle des cas d'utilisation

Relation de généralisation/spécialisation entre acteurs

- Un acteur A est une généralisation d'un acteur B si l'acteur A peut être remplacé par l'acteur B Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.
- La relation de généralisation est représentée par une flèche avec une extrémité triangulaire



Relation de généralisation/spécialisation - Exemple



Exemple

Le directeur des ventes est un préposé aux commandes avec un pouvoir supplémentaire : en plus de pouvoir passer et suivre une commande, il peut gérer le stock. Par contre, le préposé aux commandes ne peut pas gérer le stock.

Relation de généralisation/spécialisation entre acteurs

- Dans certaines situations, il arrive que deux acteurs, ou plus, présentent des similitudes dans leurs relations aux cas d'utilisation
- On peut l'exprimer en créant un acteur généralisé, éventuellement abstrait, qui modélise les aspects communs aux différents acteurs concrets. Cette relation se traduit par le concept d'héritage dans les langages orientés objet

Relation de généralisation/spécialisation

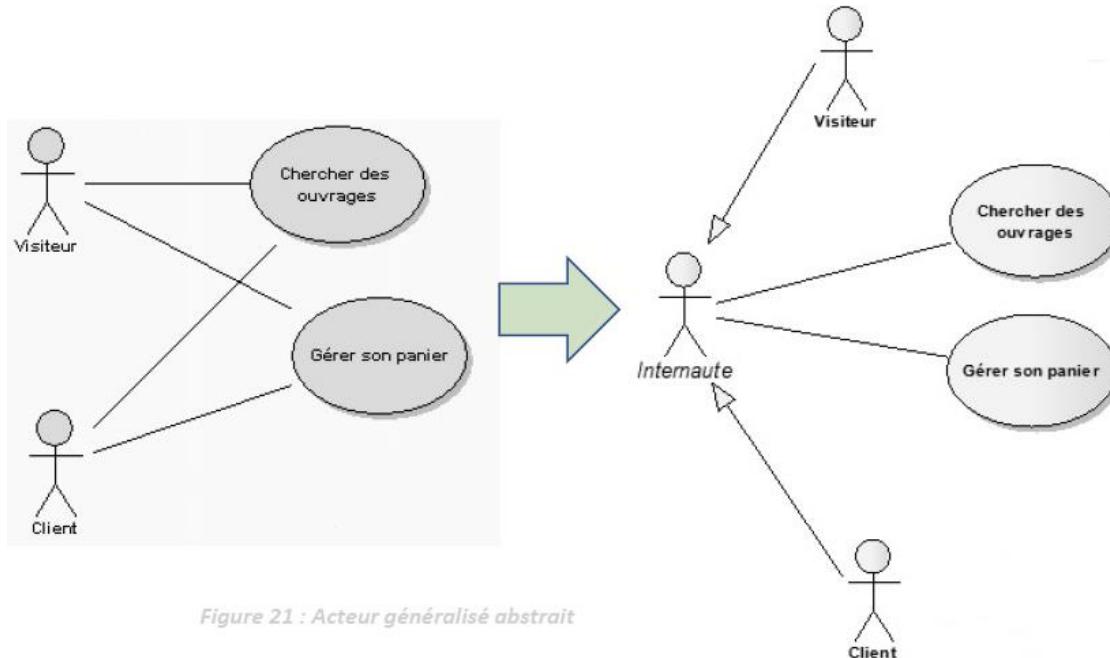


Figure 21 : Acteur généralisé abstrait

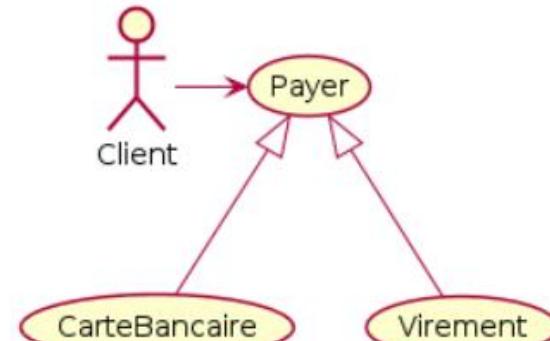
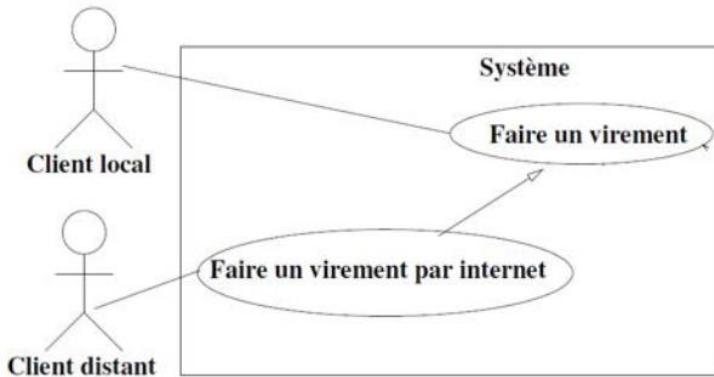


Bonne Pratique

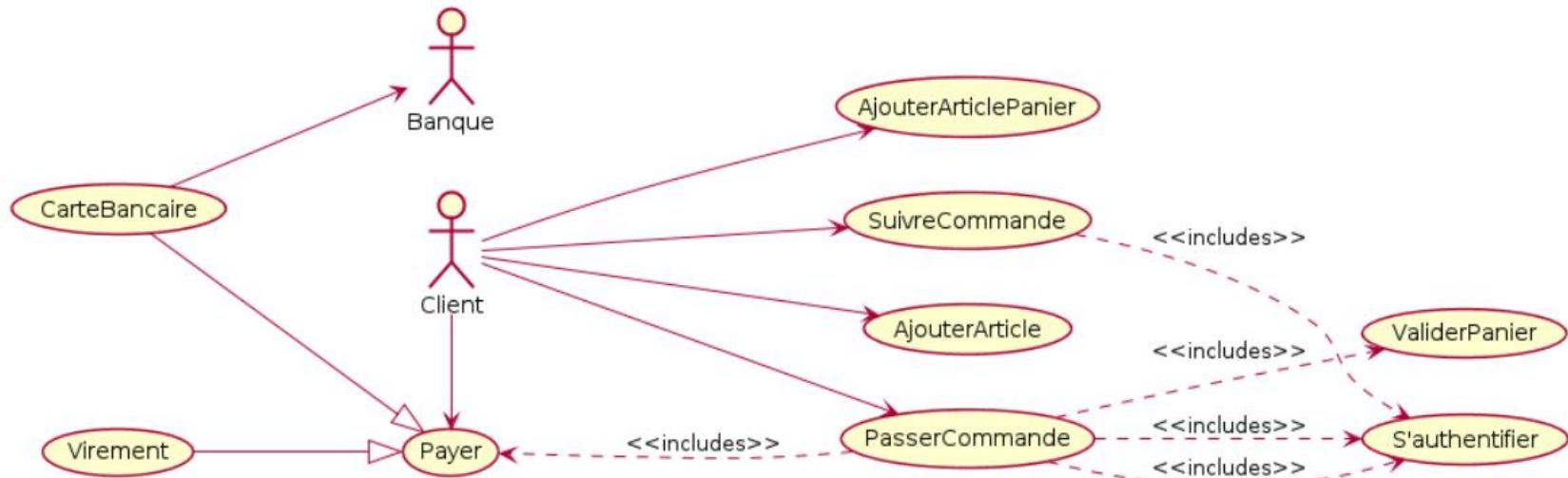
- Les deux acteurs principaux Client et Visiteur partagent deux cas d'utilisation *Chercher des ouvrages* et *Gérer son panier*
- Dans l'exemple, l'acteur Internaute est la généralisation abstraite des rôles Visiteur et Client.

Relation de généralisation/spécialisation entre cas d'utilisation

La généralisation est utilisée pour montrer qu'un cas d'utilisation est une spécialisation d'un autre cas d'utilisation. Cela signifie que le cas d'utilisation spécialisé hérite des caractéristiques du cas d'utilisation général, et peut ajouter ou modifier certains aspects de celui-ci.



Exemple complet d'un diagramme de cas d'utilisation





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces

- La modélisation par le diagramme de cas d'utilisation, comme expliqué dans le chapitre précédent, donne une description fonctionnelle du système du point de vue des **acteurs**. Elle décrit dynamiquement les scénarios d'exécution du système.
- En revanche, la conception orientée objet se concentre principalement sur une description **structurelle statique** du système à réaliser, sous forme d'un ensemble de **classes logicielles**, parfois regroupées en packages.
- **Un diagramme de classes** est un type de **diagramme structurel** utilisé dans la modélisation orientée objet, notamment avec UML (Unified Modeling Language). Il fournit une vue abstraite et statique du système en modélisant les **classes**, leurs **attributs**, leurs **méthodes**, ainsi que les **relations** entre elles.

03 - Modéliser les données du projet par un diagramme de classes

Définition du diagramme de classes

- Le diagramme de classes joue un rôle fondamental dans cette approche : il est le **diagramme le plus important** dans la modélisation orientée objet, souvent considéré comme obligatoire.
- Il fournit une représentation abstraite des objets du système qui interagissent pour réaliser **les cas d'utilisation** définis.
- Un avantage du diagramme de classes est qu'il modélise les classes du système indépendamment d'un langage de programmation particulier.

Modéliser la structure interne du système

Diagramme de classes d'Analyse (DCA)

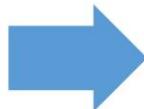


Diagramme de classes de Conception (DCC)

- représenter les **classes métiers (modèle)** issues du Cahier de Charges
- **Exemple d'un système de gestion de commandes** : classes Client, Commande, Produit

Niveau Analyse

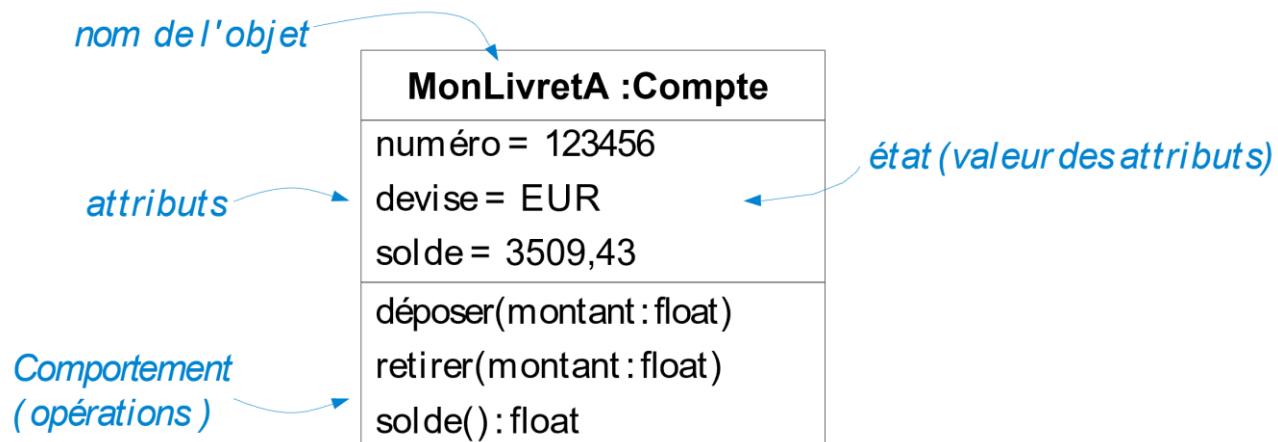
- Compléter par les **classes de Vue, de Contrôle, etc.**
- Exemple : classes pour les interfaces graphiques de l'application, classe Contrôleur pour contrôler la saisie des données, la sécurité, etc.

Niveau Conception

Structuration orienté objet - Qu'est ce qu'un objet?

Objet = Etat + Comportement + Identité

En programmation orientée objet (POO), un objet se caractérise par trois caractéristiques fondamentales : son **état** (attributs), son **comportement** (opérations), et son **identité** (adresse mémoire).

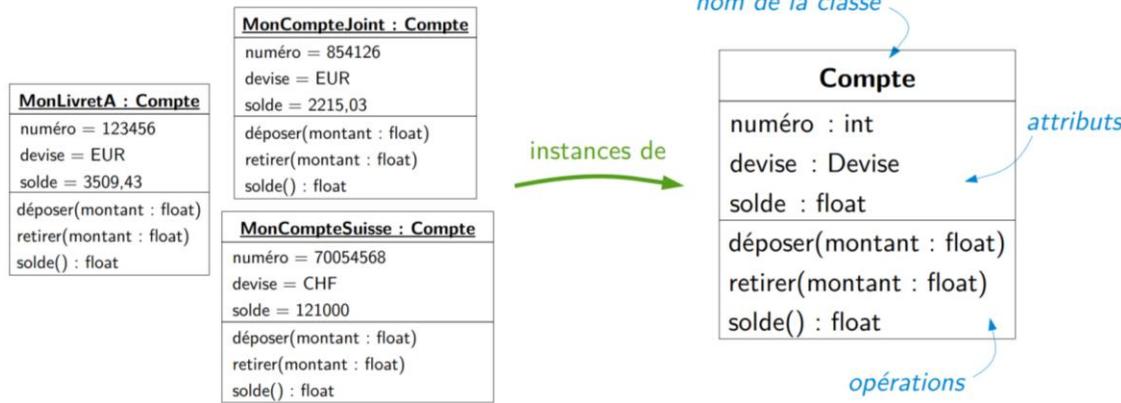


Structuration orienté objet - Qu'est ce qu'une classe ?

Qu'est-ce qu'une classe ?

Une **classe** est une représentation abstraite d'un ensemble d'objets qui partagent les mêmes caractéristiques. Elle décrit les attributs (données ou propriétés) et les méthodes (comportements ou actions) qui définissent un objet.

Objet = instance d'une classe



Représentation graphique d'une classe avec UML

Dans UML, une classe est représentée par un rectangle divisé en trois parties :

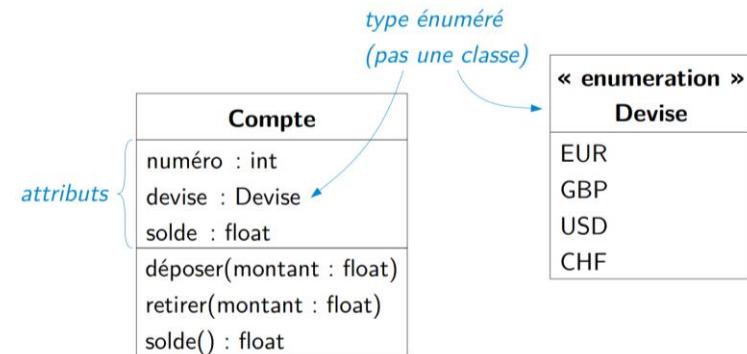
- **Nom de la classe** (en haut).
- **Attributs** (variables ou propriétés de la classe).
- **Opérations** (actions que la classe peut effectuer).

Les relations entre les classes, telles que **l'association**, **l'héritage**, ou **la composition**, sont également représentées graphiquement pour montrer comment ces classes interagissent entre elles.

Nom_de_la_classe
-attribut_1: type1
-attribut_2: type2
+opération_1(): type1
+opération_2(): void

Attributs

- **Caractéristique partagée** par tous les objets de la classe
- Associe à chaque objet une **valeur**
- Type associé simple (int, bool...), primitif (Date) ou énuméré (Une liste de valeurs possibles prédéfinies pour un attribut. L'énumération est représentée comme une classe spéciale avec un stéréotype **<>enumeration**, et ses valeurs possibles sont listées à l'intérieur de cette classe.

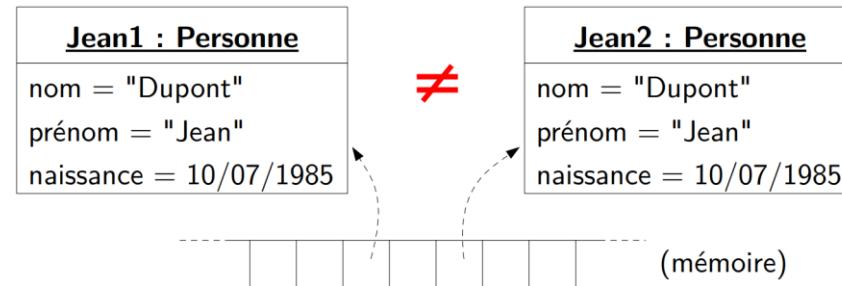


Attributs

- Caractéristique partagée par tous les objets de la classe
- Associe à chaque objet une valeur
- Type associé simple (int, bool...), primitif (Date) ou énuméré

Valeur des attributs : État de l'objet

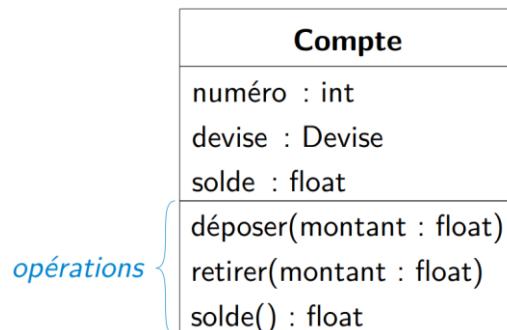
- Objets différents (identités différentes) peuvent avoir mêmes attributs



Opérations

- Service qui peut être demandé à tout objet de la classe
- **Comportement commun** à tous les objets de la classe

 Ne pas confondre avec une méthode = implantation de l'opération





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces

En programmation orientée objet, la **visibilité** et l'**encapsulation** sont des concepts clés pour protéger les données et organiser l'accès aux méthodes et attributs d'une classe.

Encapsulation :

L'encapsulation consiste à restreindre l'accès direct aux **données** (attributs) d'une classe pour assurer leur intégrité. Au lieu de permettre un accès libre à ces attributs, on contrôle l'accès à travers des **méthodes** spécifiques (souvent appelées **getters** et **setters**). Cela permet de maintenir un **niveau de contrôle** sur la manière dont les données sont modifiées ou consultées.

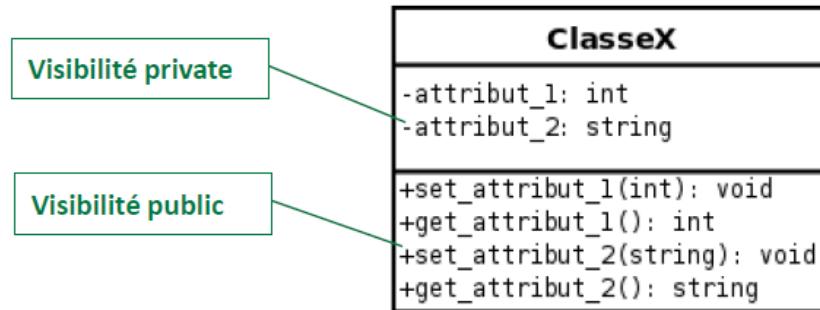
Visibilité :

La notion de **visibilité** est un aspect clé de l'encapsulation dans la programmation orientée objet. Elle permet de contrôler l'accès aux éléments d'un conteneur (comme les classes, méthodes, attributs, etc.) selon leur emplacement. En d'autres termes, la visibilité détermine **qui peut accéder** à un certain élément dans un programme.

Visibilité	Symbole	Accessible par
Public	+	Tous, aussi bien à l'intérieur qu'à l'extérieur du conteneur
Protected	#	La classe elle-même et ses sous-classes (mais pas depuis l'extérieur)
Private	-	Uniquement la classe elle-même, pas accessible par les sous-classes
Package	~	Les classes du même paquetage ou espace de noms (pas accessible de l'extérieur)

Visibilité des attributs :

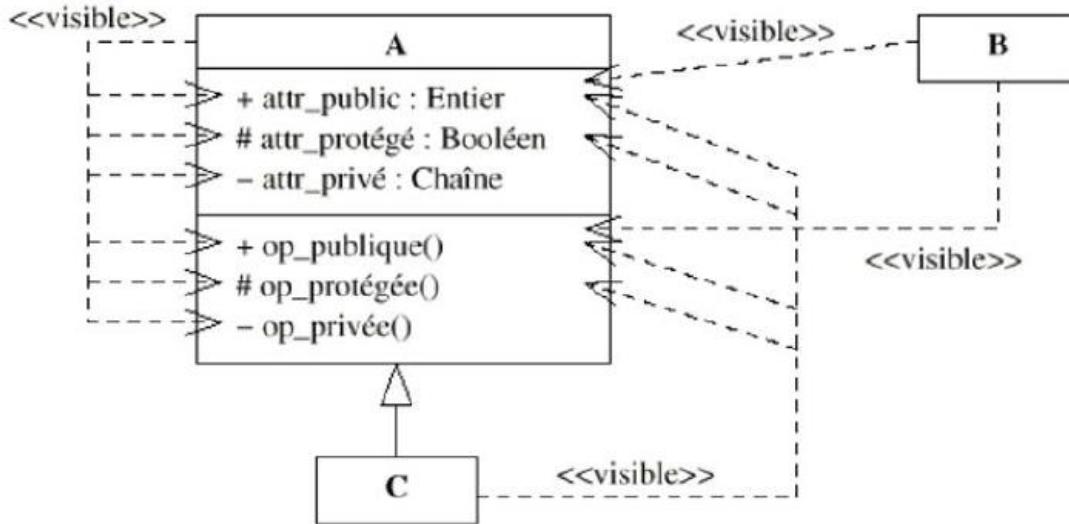
Dans la pratique, lorsque des attributs doivent être accessibles de l'extérieur, il est préférable que cet accès ne soit pas direct, mais se fasse par l'intermédiaire d'opérations (Accesseurs **getter/setter**)



Remarque

- Dans une classe, le marqueur de visibilité se situe au niveau de chacune de ses caractéristiques (attributs et opérations). Il permet d'indiquer si une autre classe peut y accéder.

Démonstration de la visibilité :





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces

Multiplicité :

La multiplicité (ou cardinalité) dans un diagramme de classe UML indique combien d'instances d'une classe peuvent être associées à une instance d'une autre classe dans une relation. Elle permet de préciser le nombre d'objets liés dans une association entre deux classes.

Nombre d'objets de la classe B associés à un objet de la classe A

Exactement 1



Au plus 1 (0 ou 1)



Au moins 1 (jamais 0)



0 ou plus





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces

Relations entre classes :

Les relations entre classes dans un diagramme de classes UML représentent la manière dont les classes interagissent entre elles. Il existe plusieurs types de relations dans UML, chacune ayant une signification et une notation spécifique.

1. Association

2. Composition

3. Héritage (Généralisation)

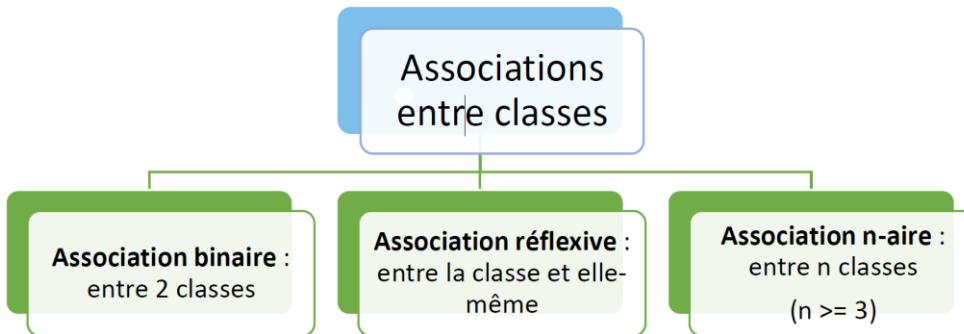
4. Dépendance

5. Réalisation

Relations d'association:

L'association est la relation la plus courante entre deux classes. Elle décrit simplement qu'une classe est liée à une autre, et qu'il peut y avoir des interactions entre leurs instances.

Il existe plusieurs types d'associations



Relations d'association:

Notation : Ligne simple avec des multiplicités aux extrémités.

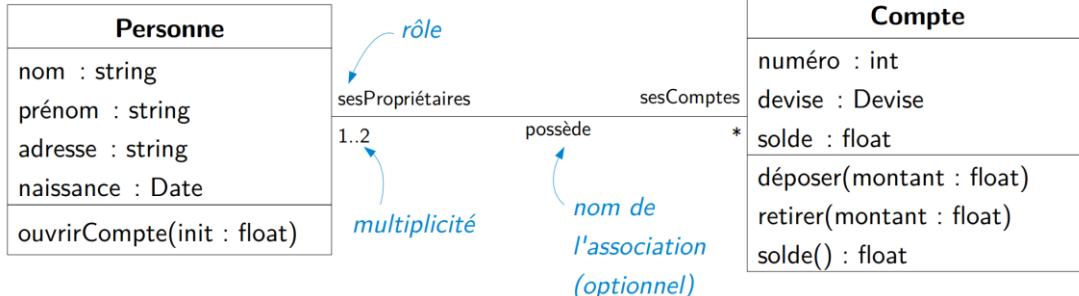
Chaque extrémité d'une association peut être nommée

Ce nom est appelé **rôle** et indique la manière dont l'objet est vu de l'autre côté de l'association.

La multiplicité (indique le nombre d'instances de classe étant en relation avec la classe situé à l'autre extrémité de l'association.

Relations d'association:

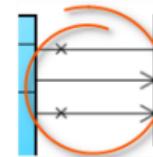
Ex: Une personne possède un ou plusieurs comptes, et un compte est associé à une ou deux personnes.



Cardinalité	Signification
0..1	Zéro ou une fois
1..1	Une et une seule fois
0..* (ou *)	De zéro à plusieurs fois
1..*	De une à plusieurs fois
m..n	Entre m et n fois
n..n (ou n)	n fois

Association entre classes – Navigabilité :

Navigabilité bidirectionnelle	Navigabilité unidirectionnelle
<ul style="list-style-type: none">• par défaut• chaque classe possède un attribut qui fait référence à l'autre classe en association.• plus complexe à réaliser• A éviter dans la mesure du possible	<ul style="list-style-type: none">• une seule classe possède un attribut qui fait référence à l'autre classe• plus fréquente• la première classe peut solliciter une deuxième et que l'inverse est impossible• peut se représenter de 3 façons différentes :<ul style="list-style-type: none">• Une croix du coté de l'objet qui ne peut pas être sollicité• Une flèche du coté de l'objet qui peut être sollicité• Les 2 représentations précédentes à la fois



Association entre classes – Navigabilité :

Orientation d'une association

Depuis un A, on a accès aux objets de B qui lui sont associés, mais pas l'inverse



Rappel : Types des attributs (simple, primitif ou énuméré)

En particulier, pas d'attribut dont le type est une classe du diagramme

Compte	
numéro : int	
devise : Devise	
solde : float	
propriétaire : Personne	*
déposer(montant : float)	
retirer(montant : float)	
solde() : float	

Mais association vers cette classe

Personne	
nom : string	
prénom : string	
adresse : string	
naissance : Date	
ouvrirCompte(init : float)	



Compte	
numéro : int	
devise : Devise	
solde : float	*
déposer(montant : float)	
retirer(montant : float)	
solde() : float	

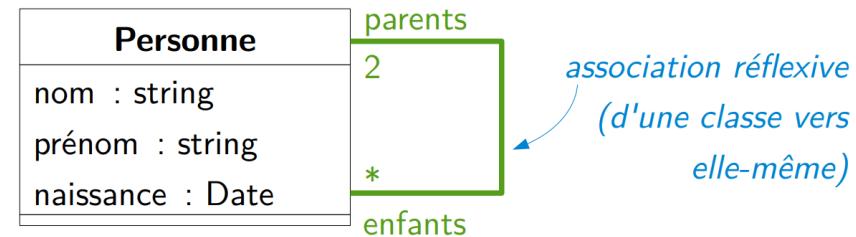
Association réflexive (ou récursive) :

Une association réflexive dans un diagramme de classe UML est une relation où une classe est associée à elle-même.

Ex: Chaque personne peut être à la fois un parent et un enfant dans la relation.

La relation ***est_parent_de*** est une association réflexive entre les instances de la classe Personne.

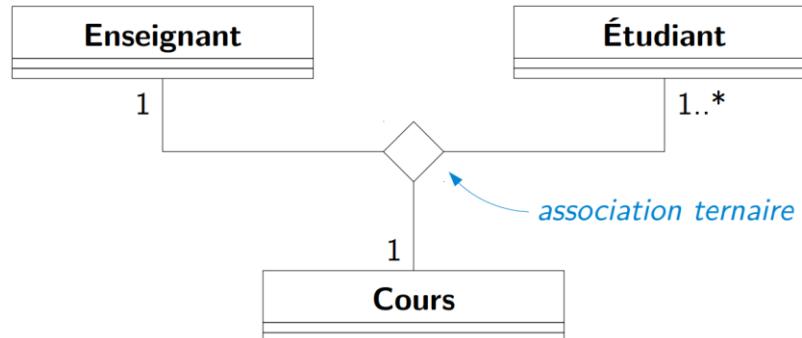
- Une personne peut avoir deux parents (père et/ou mère)
- Un parent peut avoir zéro ou plusieurs enfants.



Association n aire:

Association n aire est une association qui relie plus de deux classes. Contrairement aux associations binaires (qui relient deux classes), une association n-aire relie trois, quatre, ou plus de classes dans une relation unique.

L'association n aire se représente par un **losange** d'où part un trait allant à chaque classe

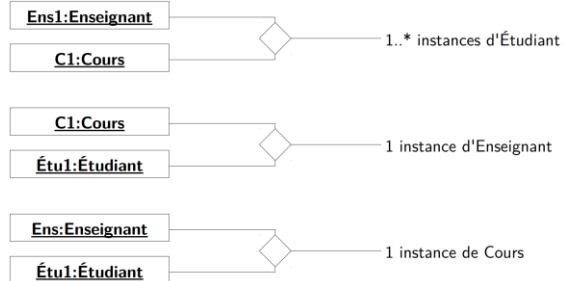
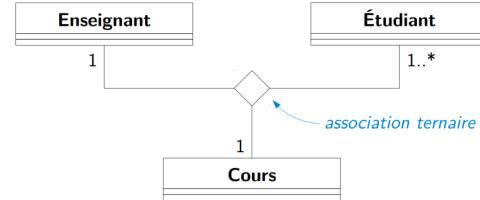


Association n aire – Lecture du diagramme :

- Un **Enseignant** donne un **seul cours**.
- Un **Cours** est donné par un **seul enseignant**.
- Un ou plusieurs **Etudiants** peuvent être inscrits à ce cours.

Exemple :

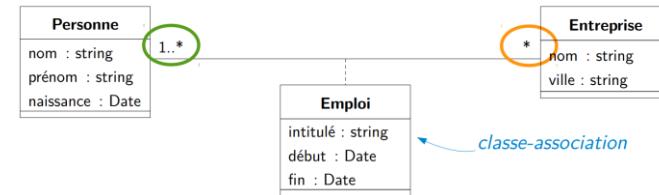
« **Mme Imane** » enseigne le Cours **M201**, et plusieurs étudiants (**Rouqaya et Ayman...**) suivent ce cours.



Classe-association :

Une classe-association en UML est utilisée pour ajouter des informations supplémentaires à une association entre deux ou plusieurs classes.

Ex: Ce diagramme représente une classe-association en UML entre trois classes : **Personne**, **Entreprise**, et **Emploi**. Il décrit la relation entre les personnes qui occupent des emplois dans des entreprises, en ajoutant des informations supplémentaires sur la relation entre une personne et une entreprise grâce à la classe-association **Emploi**.

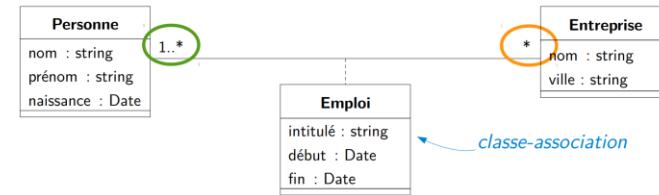


Classe-association :

La classe **Emploi** capture les détails spécifiques de la relation entre une Personne et une Entreprise.

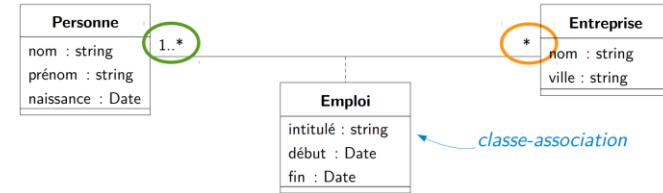
C'est une association ternaire qui donne des informations comme :

- intitulé : Le poste occupé par la personne dans l'entreprise.
- début : La date de début de l'emploi.
- fin : La date de fin de l'emploi (si applicable).



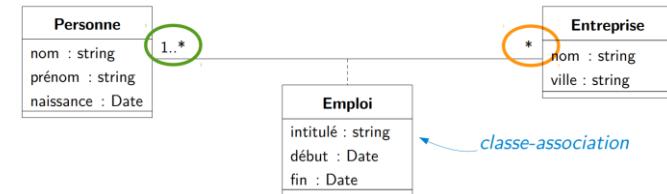
Classe-association – Lecture du diagramme :

- Une Personne peut travailler pour une ou plusieurs entreprises et occuper différents emplois à travers sa carrière (multiplicité 1..*).
- Une Entreprise peut employer plusieurs personnes (multiplicité *), chacune dans un emploi spécifique.
- La classe-association **Emploi** relie ces deux entités et contient des informations supplémentaires concernant la relation d'emploi, comme le poste occupé, la date de début, et la date de fin.



Classe-association – Exemple concret :

- **Personne A** travaille comme **Ingénieur** dans l'**Entreprise X** depuis le 01-01-2020.
- **Personne B** travaille comme **Manager** dans la même **Entreprise X** depuis le 01-06-2021.
- Les informations sur chaque emploi (intitulé, début, fin) sont stockées dans la classe-association **Emploi**.



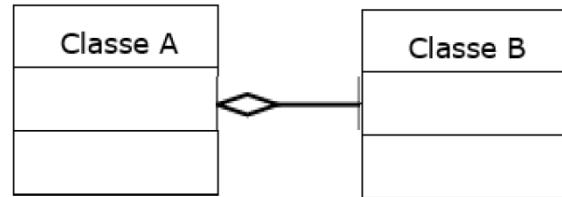
Agrégation

L'agrégation indique qu'un objet A possède un autre objet B

mais contrairement à la composition, l'objet B peut exister
indépendamment de l'objet A.

La suppression de l'objet A n'entraîne pas la suppression de
l'objet B.

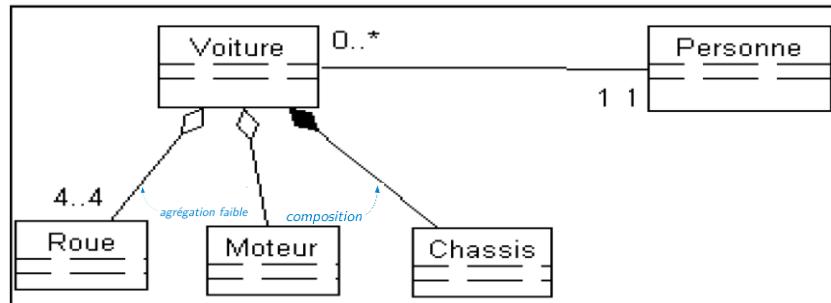
Elle se représente par un **losange vide** du côté de l'objet
conteneur.



Agrégation – Exemple concret :

Une **voiture** est composée de **4 roues et un moteur**. Ils peuvent être utilisés pour une autre voiture.

Par exemple, si une Voiture est détruite, ses Roues et son Moteur peuvent être réutilisés sur une autre voiture, vendues séparément, ou stockées. Elles ne dépendent pas de l'existence de la voiture pour exister.



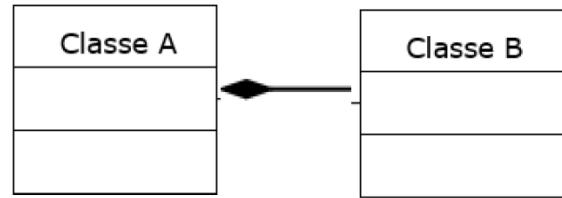
Composition

La composition indique qu'un objet A (appelé conteneur) est constitué d'un autre objet B.

Cet objet A n'appartient qu'à l'objet B et ne peut pas être partagé avec un autre objet.

C'est une relation très forte si l'objet A disparaît, alors l'objet B disparaît aussi.

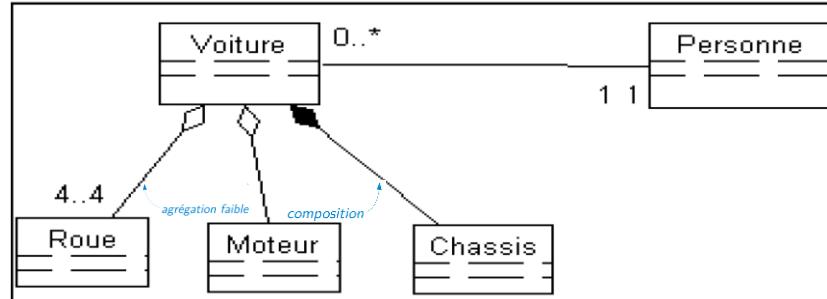
Elle se représente par un **losange plein** du côté de l'objet conteneur.



Composition – Exemple concret :

Une **voiture** est composée d'un **châssis**. Il ne peut pas être utilisé pour une autre voiture.

Si la Voiture A est détruite dans un accident, le Châssis est également détruit.



Relation d'héritage:

La **relation d'héritage** en UML (aussi appelée généralisation) représente une relation de type "**est-un**" entre une classe **parente** (ou **super-classe**) et une ou plusieurs classes **enfant** (ou **sous-classes**). L'idée derrière l'héritage est que la sous-classe hérite des attributs et des méthodes de la super-classe, tout en pouvant ajouter ou spécialiser son propre comportement.

Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence

Spécialisation : raffinement d'une classe en une sous-classe.

Généralisation : abstraction d'un ensemble de classes en super-classe.

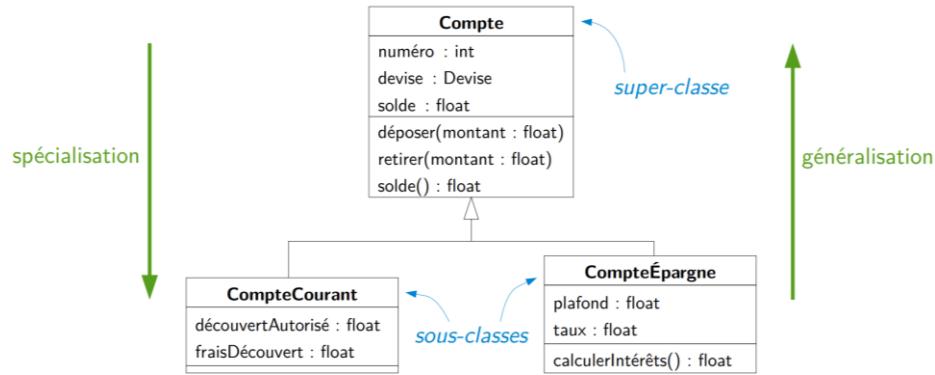
CompteCourant	CompteEpargne
numéro : int devise : Devise solde : float découvertAutorisé : float fraisDécouvert : float déposer(montant : float) retirer(montant : float) solde() : float calculerIntérêts() : float	numéro : int devise : Devise solde : float plafond : float taux : float déposer (montant : float) retirer(montant : float) solde() : float calculerIntérêts() : float

Relation d'héritage:

Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence.

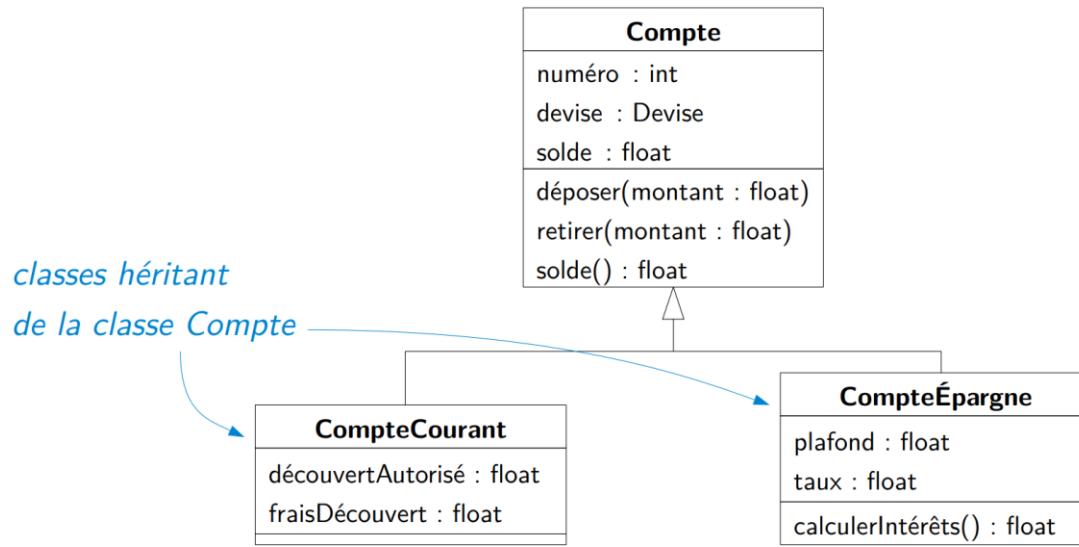
Spécialisation : raffinement d'une classe en une sous-classe.

Généralisation : abstraction d'un ensemble de classes en super-classe.



Relation d'héritage:

Héritage : Construction d'une classe à partir d'une classe plus haute dans la hiérarchie (partage des attributs, opérations, contraintes...)





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces

Classes concrètes et abstraites

Classe concrète

- Possède des instances.
- Elle constitue un modèle complet d'objet
- Tous les attributs et méthodes sont complètement définis

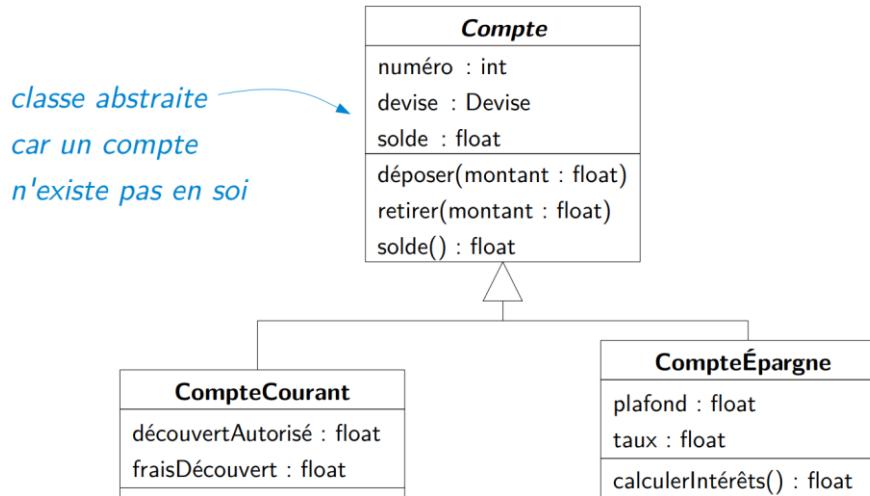
Classes abstraites

- Ne peut pas posséder d'instance directe
- Elle ne fournit pas une description complète
- Elle force le passage par l'héritage pour avoir des sous-classes concrètes
- Elle sert à factoriser des attributs et des méthodes à ses sous-classes
- Une classe abstraite possède généralement des méthodes communes non définies

Classes concrètes et abstraites

Classe sans instance, seulement une base pour classes héritées

Notation : nom de la classe en italique (ou stéréotype « abstract »)





03 - Modéliser les données du projet par un diagramme de classes

1. Définition du diagramme de classes
2. Visibilité, encapsulation des méthodes et attributs
3. Multiplicité (cardinalité)
4. Relations entre classes
5. Classes concrètes et abstraites
6. Relation de réalisation et notion d'interfaces