SUP MTI
École Supérieure de Management, de Télécommunication et d'Informatique

UML: Analyse et conception OO

Pr. Mariame Nassit

Plan

1. Introduction à la Modélisation Orienté Objet
    1. Introduction
    2. Le génie logiciel
    3. Modélisation
2. Méthode objet élémentaire avec UML
    1. Le diagramme du cas d'utilisation
    2. Le diagramme de classe
    3. Le diagramme d'objet
    4. Le diagramme de séquence
    5. Le diagramme d'activité

Partie 1

Introduction à la Modélisation Orienté Objet

Introduction

Introduction

Contextes

- La compréhension d'une problématique complexe, comme le développement d'applications, repose de plus en plus sur l'utilisation de la modélisation informatique.
- L'accélération du renouvellement des technologies, combinée à la pression économique et concurrentielle qui pèse sur les entreprises, contraint les acteurs du monde informatique à développer des solutions toujours plus rapidement, tout en assurant une amélioration continue de la qualité et des performances des systèmes d'information.
- Internet a largement contribué au développement de nombreuses applications, dont une grande partie repose sur des solutions utilisant des langages de programmation orientés objet tels que Java, C++ et C#.

Introduction

Objectif

- Depuis longtemps, le langage UML (*Unified Modelling Language*) a été conçu pour formaliser les besoins des clients et les traduire en programmes concrets.
- Il constitue un outil de communication efficace entre les modélisateurs, les informaticiens et les autres métiers, en dépassant les barrières des disciplines et des aspects techniques liés à l'informatique.
- UML a naturellement comblé le manque d'une approche méthodologique pour les concepteurs et développeurs, leur permettant de formaliser l'analyse et la conception technique des logiciels.

Introduction

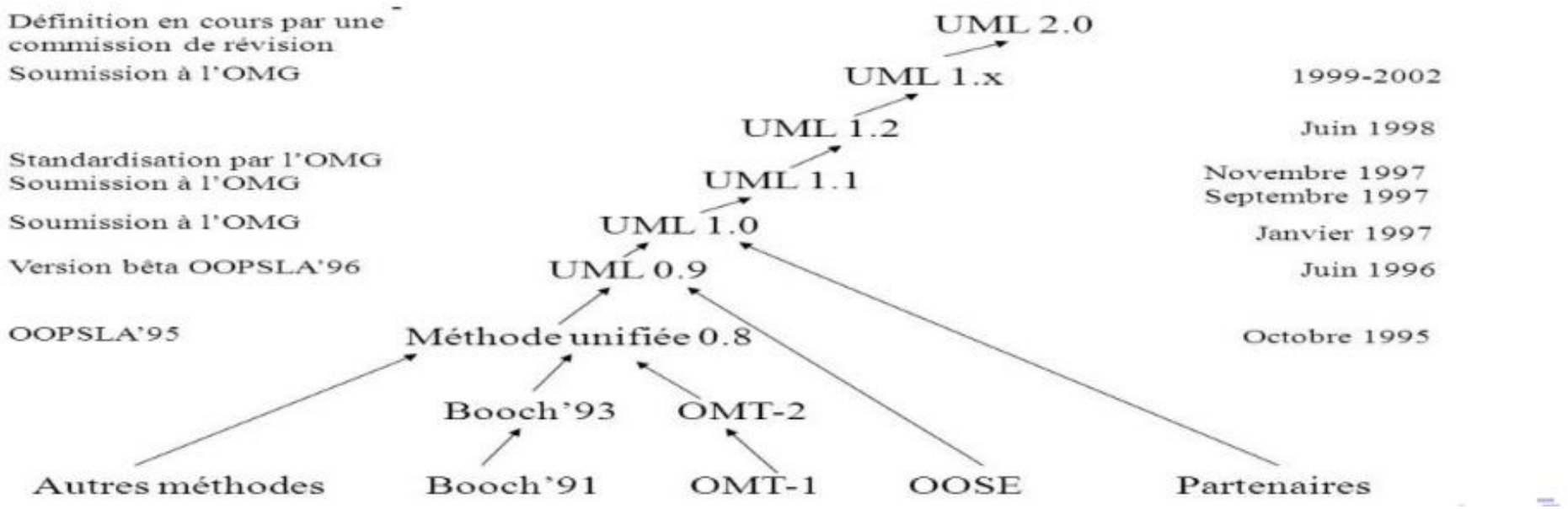
Spécification

- Depuis plus de 30 ans, le modèle entité-association est utilisé pour concevoir des bases de données. Ce modèle est fiable et encore largement employé par les outils informatiques aujourd'hui.
- Ces dernières années, la notation UML est devenue un outil essentiel pour modéliser et développer des applications utilisant des langages orientés objet comme C++ et Java.
- Même si UML n'était pas conçue à l'origine pour les bases de données, elle offre un formalisme adapté aux concepteurs d'objets métiers et de bases de données.
- Aujourd'hui, UML est un langage graphique de modélisation qui répond à un vrai besoin. Elle est devenue un standard reconnu, basé sur une norme claire et structurée.

Introduction

Histoire de l'UML

- UML s'inspire principalement des méthodes orientées objet développées par Booch, OMT (*Object Modeling Technique*) et OOSE (*Object Oriented Software Engineering*), créées respectivement par Grady Booch, Rumbaugh et Jacobson.
- UML est le résultat de la fusion des approches OOA, OOSE et OMT.
- Adopté comme standard par l'OMG en novembre 1997, UML est devenu une référence incontournable.



Le génie logiciel

Le génie logiciel

L’informatisation

- **L'informatisation** est un phénomène majeur de notre époque.
- Elle est présente dans :

  Les objets du quotidien eux-mêmes.

  Les processus de **conception** et de **fabrication** de ces objets.
- **L'informatique** joue un rôle central dans les grandes entreprises.
- Le **système d'information** d'une entreprise comprend :

  Matériels (20 % des investissements).

  Logiciels (80 % des investissements).
- Depuis quelques années :

  Peu de fabricants produisent le matériel, devenu **fiable** et **standardisé**.
- Les principaux problèmes en informatique concernent essentiellement les **logiciels**.

Le génie logiciel

Les logiciels

- Un logiciel ou une application est un ensemble de programmes permettant à un ordinateur d'accomplir une tâche précise (ex. : comptabilité, gestion des prêts).
- Développement des logiciels :
  - Petits logiciels : développés par une personne ou une petite équipe.
  - Grands logiciels : nécessitent de grandes équipes, posant des problèmes de conception et coordination.
  - Le développement est crucial : il représente l'essentiel du coût et détermine la réussite d'un produit.
- Étude du Standish Group (échantillon : 365 entreprises, 8 380 applications) :
  - 16,2 % des projets respectent les prévisions initiales.
  - 52,7 % dépassent les coûts et délais (facteur 2 à 3) avec des fonctionnalités réduites.
  - 31,1 % sont abandonnés en cours de développement.

Le génie logiciel

Les logiciels

- Pour les grandes entreprises :
  - **9 %** de succès seulement.
  - **37 %** des projets sont arrêtés.
  - **50 %** aboutissent avec des dépassements de coûts et de délais.
- Causes des échecs :
  - Principalement dues à la **maîtrise d'ouvrage** (le client), pas à l'informatique.
- Solution :

- Le développement professionnel suit des règles strictes pour :

  Encadrer la conception.

  Faciliter le travail en groupe.

  Assurer la maintenance du code.

- Naissance du génie logiciel : discipline permettant de structurer le développement des logiciels.

Le génie logiciel

Les logiciels

- Définition du génie logiciel :

Un ensemble de méthodes, techniques et outils consacrés à la conception, au développement et à la maintenance des systèmes informatiques.

- Focus du génie logiciel :
  - Spécification et production du **code source**.
  - Gestion du **cycle de vie des logiciels** :
    - Analyse du besoin.
    - Élaboration des spécifications.
    - Conceptualisation.
    - Développement.
    - Tests.
    - Maintenance.

Le génie logiciel

Les objectifs du génie logiciel

Pour créer des logiciels de grande taille, il est essentiel de respecter :

- **Coûts :** Les coûts prévus pour la réalisation doivent être respectés.
- **Qualité :** La qualité d'un logiciel désigne sa capacité à répondre efficacement aux besoins pour lesquels il a été conçu. Elle repose sur plusieurs critères (Validité, fiabilité, compatibilité, ...)
- **Fonctionnalités :** Les spécifications doivent répondre aux besoins réels du client.
- **Délais :** Les délais fixés ne doivent pas être dépassés.

Le génie logiciel

Notion de qualité pour un logiciel

- Les facteurs de qualité dépendent du domaine d'application et des outils utilisés. Parmi les principaux facteurs :
  - Validité : Capacité du logiciel à remplir ses fonctions selon le cahier des charges.
  - Fiabilité (Robustesse) : Fonctionnement même dans des conditions anormales.
  - Extensibilité (Maintenance) : Facilité de modification ou d'ajout de nouvelles fonctions.
  - Réutilisabilité : Possibilité de réutiliser tout ou partie du logiciel pour de nouvelles applications.
  - Compatibilité : Facilité d'intégration avec d'autres logiciels.
  - Efficacité : Utilisation optimale des ressources matérielles.
  - Portabilité : Capacité à fonctionner sur différents environnements matériels et logiciels.

Notion de qualité pour un logiciel

- Les facteurs de qualité dépendent du domaine d'application et des outils utilisés. Parmi les principaux facteurs :
  - Vérifiabilité : Simplicité de préparation des tests.
  - Intégrité : Protection du code et des données contre les accès non autorisés.
  - Facilité d'emploi : Simplicité d'apprentissage, d'utilisation, de gestion des erreurs et de correction en cas d'erreur.

Remarque :

- Ces facteurs peuvent parfois être contradictoires. Les compromis doivent être choisis en fonction du contexte du projet.

Le génie logiciel

Les phases de réalisation d'un logiciel

- La réalisation d'un logiciel comprend 7 phases :

1. **Analyse** : Recueil des besoins et des attentes des utilisateurs.
2. **Conception** : Élaboration de l'architecture et des fonctionnalités du logiciel.
3. **Réalisation** : Codage et développement du logiciel.
4. **Mise en place** : Installation et déploiement du logiciel dans son environnement.
5. **Exploitation** : Utilisation du logiciel en production, avec les ajustements, corrections et évolutions nécessaires.
6. **Maintenance** : Adaptations, corrections et améliorations continues pour garantir le bon fonctionnement du logiciel et répondre aux nouveaux besoins.
7. **Démontage** : Désinstallation ou arrêt définitif du logiciel.

Les phases d'analyse

- La phase d'analyse est une étape cruciale dans la réalisation d'un logiciel, car elle pose les bases du projet. Elle permet de comprendre les besoins des utilisateurs et de définir les objectifs du logiciel. Voici les principales étapes et actions de cette phase :
  - **Recueil des besoins** :
    - **Objectif** : Identifier précisément ce que les utilisateurs attendent du logiciel.
    - **Méthodes** : Interviews avec les utilisateurs, enquêtes, observation, analyse des documents existants (cahiers des charges, spécifications, etc.).
    - **Résultat** : Un document de spécifications fonctionnelles détaillant les besoins et exigences des utilisateurs.
  - **Analyse des exigences** :
    - **Objectif** : Clarifier et analyser les besoins recueillis afin de comprendre les attentes réelles.
    - **Méthodes** : Séances de travail avec les parties prenantes (clients, utilisateurs, experts), identification des contraintes techniques, légales, et financières.
    - **Résultat** : Un document d'exigences, qui peut inclure des exigences fonctionnelles (ce que le logiciel doit faire) et non fonctionnelles (performance, sécurité, etc.).

La phase d'analyse

- **Modélisation des processus métier** :
  - **Objectif** : Cartographier les processus métier existants pour mieux comprendre les flux de travail et les interactions.
  - **Méthodes** : Diagrammes de flux, modélisation UML, cartes de processus.
  - **Résultat** : Des diagrammes ou modèles représentant les processus actuels, permettant de mieux cibler les besoins du logiciel.
- **Définition des contraintes et des limites** :
  - **Objectif** : Identifier les contraintes techniques, organisationnelles, et économiques qui limiteront la solution.
  - **Méthodes** : Analyse des systèmes existants, étude des technologies disponibles, évaluation du budget et des délais.
  - **Résultat** : Un document définissant les principales contraintes qui guideront la conception et le développement du logiciel.

La phase d'analyse

- **Identification des utilisateurs et de leurs rôles** :
  - **Objectif** : Déterminer les types d'utilisateurs du logiciel et leurs interactions avec le système.
  - **Méthodes** : Création de profiles, cartographie des utilisateurs, définition des rôles et droits d'accès.
  - **Résultat** : Une liste des utilisateurs, leurs rôles et leurs attentes par rapport au logiciel.
- **Rédaction du cahier des charges :**
  - **Objectif :** Formaliser les résultats de l'analyse sous forme de cahier des charges.
  - **Méthodes :** Compilation des besoins, exigences, processus et contraintes dans un document de référence.
  - **Résultat :** Le cahier des charges, qui servira de base pour la phase de conception et le suivi du projet.

La phase d'analyse

- **Validation avec les parties prenantes :**
  - **Objectif :** S'assurer que les besoins et exigences sont correctement compris et validés par les utilisateurs et parties prenantes.
  - **Méthodes :** Réunions de validation, présentations des documents d'analyse, ajustements en fonction des retours.
  - **Résultat :** L'approbation des besoins et exigences, marquant la fin de la phase d'analyse.

Remarques:

Objectifs principaux de la phase d'analyse :

- Garantir une compréhension claire et partagée des besoins du client.
- Identifier les contraintes techniques et organisationnelles.

Le génie logiciel

La phase de conception

- La **phase de conception** est une étape cruciale dans le cycle de vie du développement d'un logiciel.
- Elle intervient après la **phase d'analyse** et précède la **phase de réalisation (codage)**.
- L'objectif principal de cette phase est de traduire les besoins fonctionnels et techniques identifiés en une architecture détaillée et organisée du futur logiciel.
- **Objectif de la phase de conception**
  - Transformer les besoins exprimés dans la phase d'analyse en une **solution technique claire et structurée**.
  - Définir **l'architecture** globale du logiciel pour assurer sa robustesse, sa performance et son évolutivité.
  - Préparer le travail des développeurs en spécifiant les **composants**, les **modules** et leur **interaction**

Le génie logiciel

La phase de conception

- **Activités principales :**

1. **Conception générale** : Découpage du logiciel en modules et sous-systèmes.
2. **Conception détaillée** : Description précise de chaque module, interfaces et algorithmes.
3. **Modélisation** : Création de diagrammes (UML) pour représenter les composants et les flux de données.
4. **Design de l'interface utilisateur** : Définition des écrans et interactions.

La phase de conception

Exemple simple de conception

Projet : Création d'une application pour une bibliothèque.

- **Conception générale** :
  - Modules principaux : Gestion des utilisateurs, Gestion des livres, Gestion des emprunts.
- **Conception détaillée** :
  - **Gestion des utilisateurs** : Ajouter, modifier, supprimer un utilisateur.
  - **Gestion des livres** : Ajouter un livre, consulter la disponibilité, supprimer un livre.
  - **Gestion des emprunts** : Enregistrer un prêt, gérer les retours, afficher les retards.

Le génie logiciel

La phase de réalisation

- Une fois la conception finalisée et approuvée, la réalisation du logiciel doit être mise en œuvre.
- Cette étape consiste à convertir la conception du logiciel en un produit opérationnel (exécutable).
- L'objectif de cette phase est d'assurer la mise en production et l'utilisation du logiciel.

Le génie logiciel

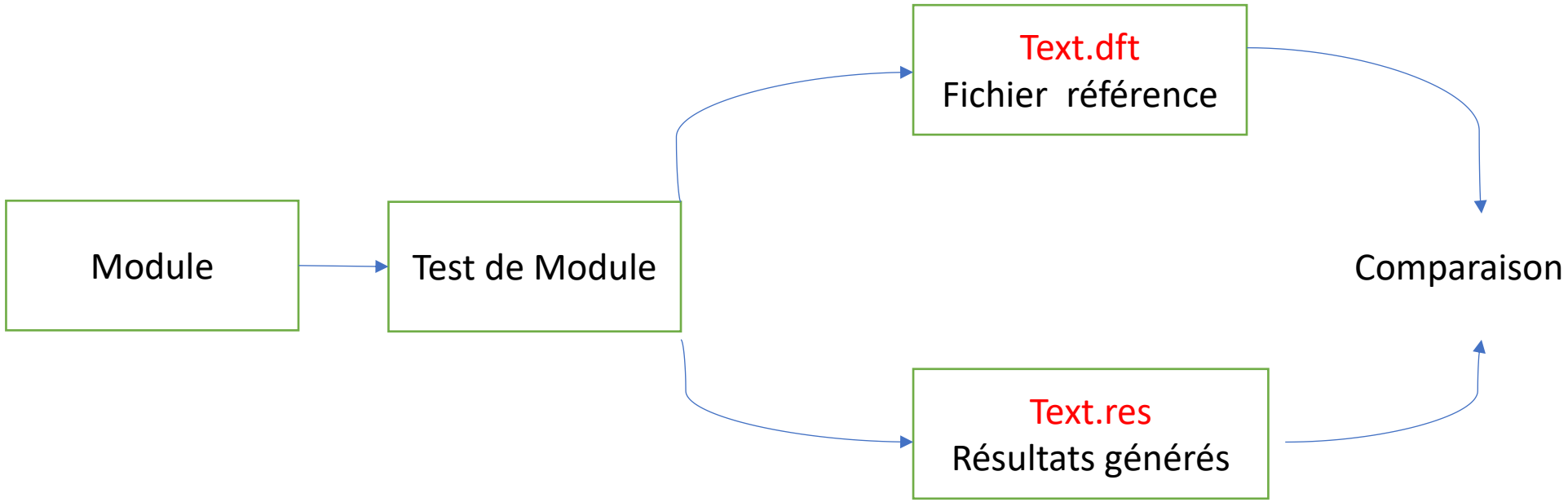
La phase de réalisation

- **Spécification :**
  La spécification consiste à définir toutes les exigences du logiciel :
  - **Fonctionnelles** : Ce que le logiciel doit accomplir (sous-programmes, traitements à réaliser).
  - **Non fonctionnelles** : Contraintes liées à l'environnement, la performance et la sécurité.
- **Conception :**
  La conception vise à fournir une solution en réponse aux spécifications :
  - **Architecture** : Organisation des modules, objets, classes et fonctions.
  - **Matériel** : Définition de l'environnement de développement et des applications externes utilisées.
- **Implantation (ou codage) :**
  Cette étape consiste à traduire les spécifications et la conception en code informatique.

Le génie logiciel

La phase de réalisation

- La validation permet de s'assurer du bon fonctionnement de chaque module.



Le génie logiciel

La phase de mise en place

- Une fois le logiciel réalisé (avec l'exécutable final obtenu), il est nécessaire de passer à sa mise en œuvre.
  Mettre en œuvre un logiciel implique son déploiement, ce qui comprend :
  - L'installation du logiciel dans l'environnement de production.
  - La préparation, dès le début de la phase de réalisation, d'une machine dédiée dans cet environnement.
- Le déploiement d'un logiciel nécessite également la configuration de l'environnement de production, notamment :
  - L'interconnexion avec tous les systèmes et processus qui doivent communiquer avec le logiciel.
  - Le transfert et la conversion des données dans un nouveau format adapté à l'usage (étape de récupération).

Le génie logiciel

La phase d’exploitation

- Une fois le logiciel installé, son exploitation devient essentielle.
- Cette phase comprend deux étapes principales qui s'étendent depuis l'installation du logiciel jusqu'à la fin de son utilisation.
  - Tout d'abord, le logiciel doit être configuré, ce qui inclut l’insertion des données nécessaires.
  - Ensuite, un traitement des données est effectué (tests du logiciel) afin d’identifier et de corriger d’éventuelles erreurs.

Le génie logiciel

La phase de maintenance

- Une fois les erreurs identifiées, il est nécessaire de procéder à leur correction, ce qui correspond à la phase de maintenance du logiciel.
  La maintenance doit être effectuée de manière régulière et se divise en trois types principaux :
  - **Maintenance corrective** : Elle vise à corriger les erreurs détectées pendant l'exploitation, notamment les bugs identifiés.
  - **Maintenance adaptative** : Elle consiste à adapter le logiciel à de nouveaux besoins ou à un environnement spécifique, en tenant compte des demandes des utilisateurs pour améliorer leur efficacité au travail.
  - **Maintenance perfective** : Lorsque le logiciel répond à toutes les exigences initiales du cahier des charges, cette maintenance se concentre sur son amélioration et son évolution. L'objectif est d'optimiser ses performances, par exemple en instaurant un système de suivi des bugs (incluant leur date, les conditions de reproduction, les solutions et les corrections apportées).

Le génie logiciel

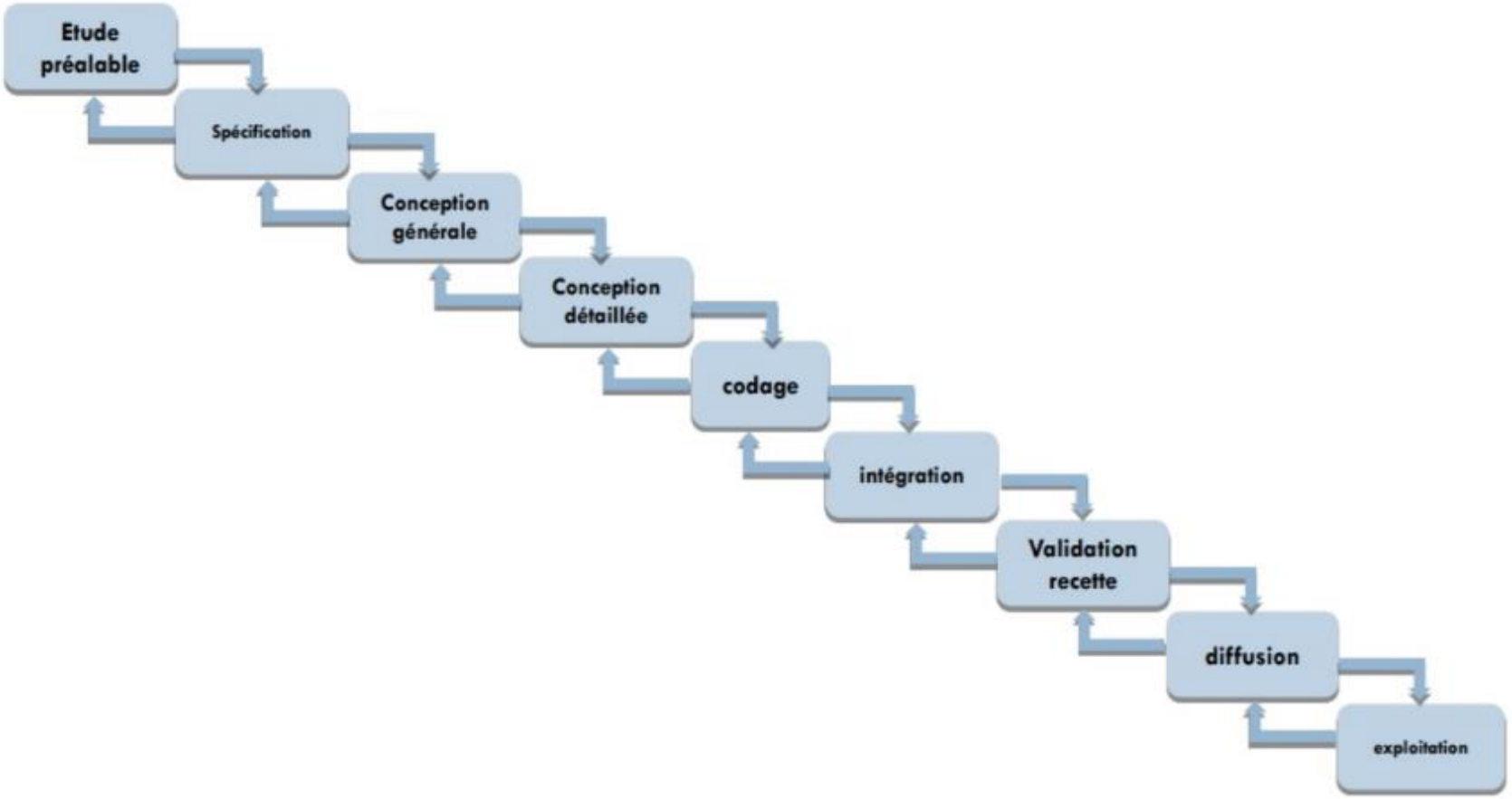
La phase de démontage

- Malgré toutes les maintenances effectuées, un logiciel peut atteindre un point où il ne répond plus aux besoins du client.
- Il est alors considéré comme révolue, marquant la fin de sa vie, un processus appelé démontage.
- Un logiciel est déclaré révolue dans les cas suivants :
  - Il ne répond plus aux attentes des utilisateurs, et aucune maintenance ne peut résoudre ce problème.
  - Il est dépassé technologiquement.
  - Le client n'en a plus l'utilité.
- Le démontage correspond à la fin de vie du logiciel. Cette étape implique la suppression complète des exécutables, des pilotes et des données associées.

Le génie logiciel

Autres modèles de développement logiciel

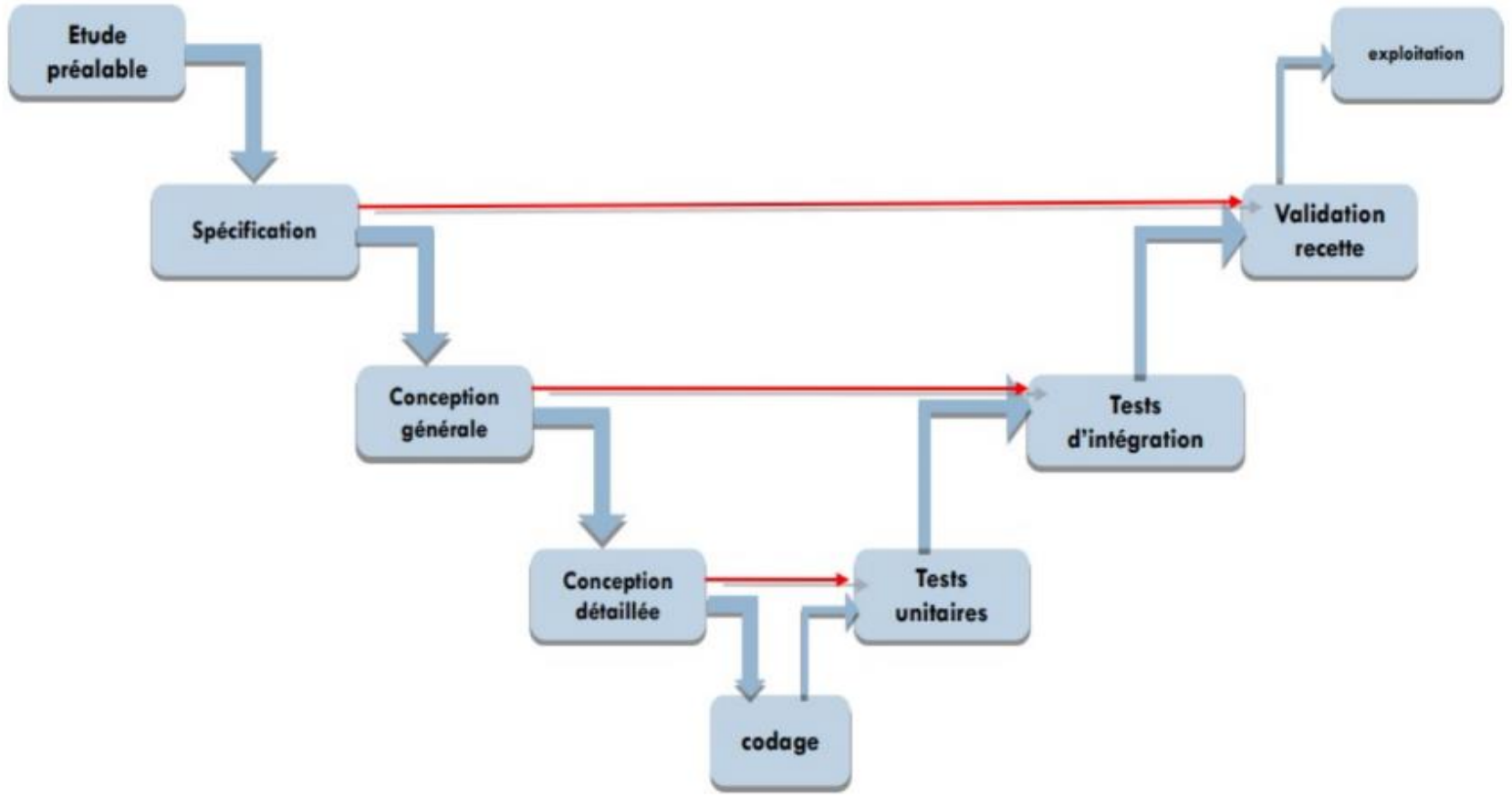
❑ Le modèle en cascade



Le génie logiciel

Autres modèles de développement logiciel

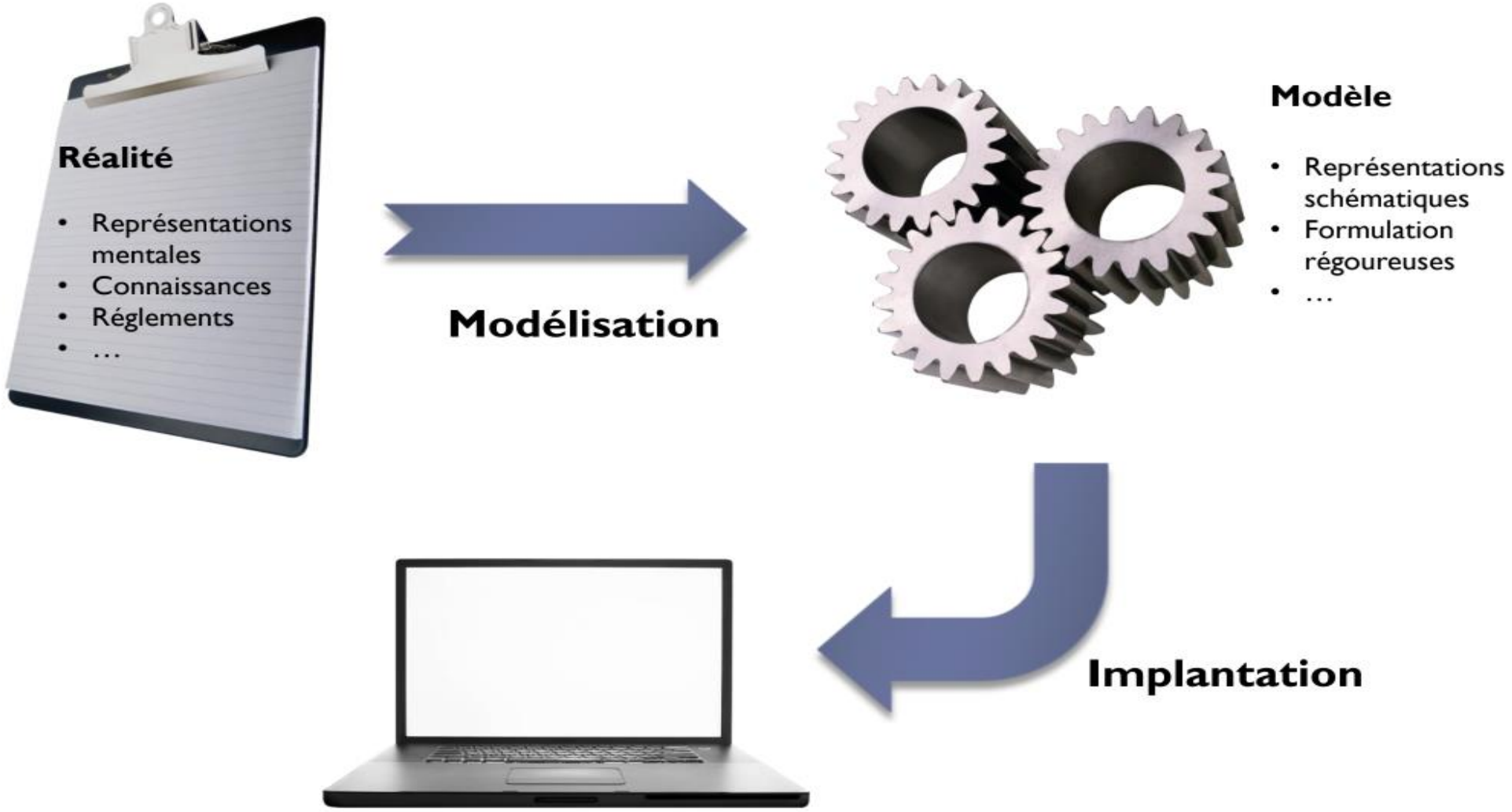
❑ Le modèle en V



Modélisation

Modélisation

Modélisation



Modélisation

Modèle

- Un modèle est une représentation simplifiée de la réalité, conçue pour omettre certains détails du monde réel.
  - Il sert à réduire la complexité d'un phénomène en écartant les éléments qui n'affectent pas significativement son comportement.
  - Il reflète les aspects jugés essentiels par le concepteur pour comprendre et prédire le phénomène étudié. Les limites d'un modèle sont déterminées par les objectifs pour lesquels il a été créé.

Modélisation

Langage de modélisation

- Un langage de modélisation doit définir :
  - La sémantique des concepts ;
  - Une notation pour représenter ces concepts ;
  - Des règles pour leur construction et leur utilisation.
- Il existe différents types de langages, selon leur niveau de formalisation :
  - **Langages formels** : souvent basés sur des mathématiques, ils offrent un fort pouvoir d'expression et permettent de réaliser des preuves formelles sur les spécifications.
  - **Langages semi-formels** (MERISE, UML...) : généralement graphiques, ils sont moins puissants en termes d'expression, mais plus simples à utiliser.
- L'industrie du logiciel propose une grande variété de langages de modélisation :
  - Ceux adaptés aux systèmes procéduraux (comme MERISE) ;
  - Ceux destinés aux systèmes temps réel (ROOM, SADT...) ;
  - Ceux conçus pour les systèmes orientés objets (OMT, Booch, UML...).
- Enfin, les outils tels que les Ateliers de Génie Logiciel jouent un rôle crucial pour rendre les langages de modélisation utilisables dans des contextes pratiques.

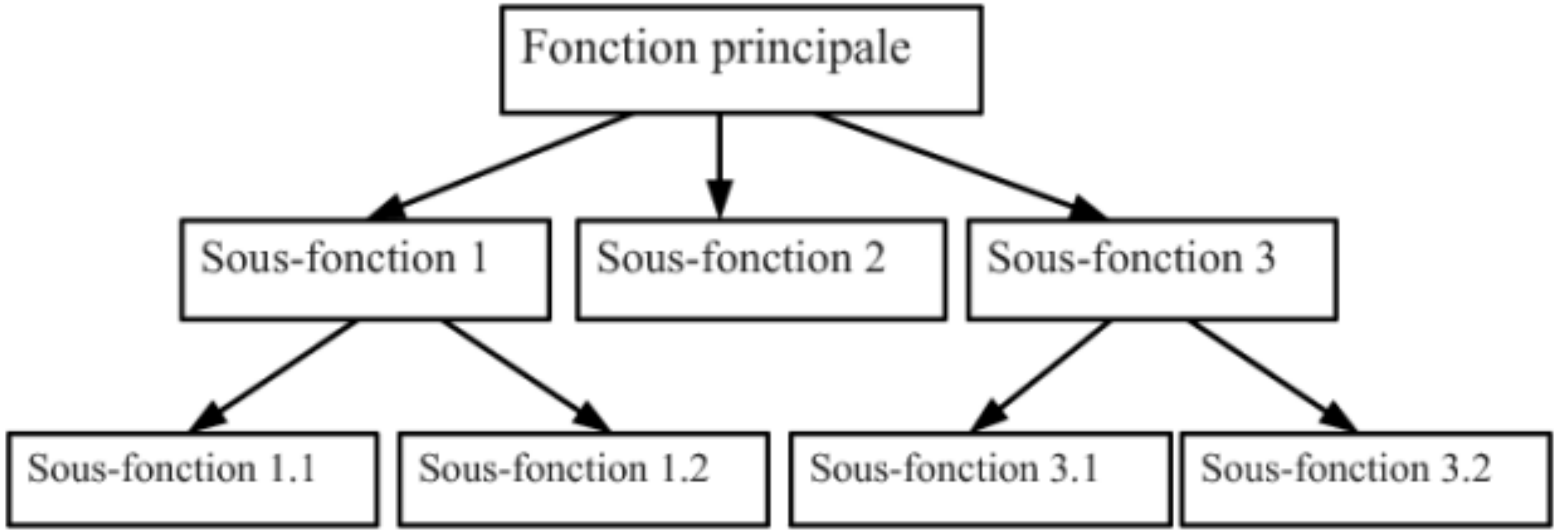
Modélisation

Modélisation par décomposition fonctionnelle

❑Approche descendante

- La **modélisation par décomposition fonctionnelle**, ou **approche descendante (top-down)**, est une méthode structurée qui consiste à diviser un système complexe en sous-systèmes plus simples et gérables.
- Cette approche est largement utilisée pour comprendre, analyser et concevoir des systèmes complexes en se concentrant sur leurs fonctions principales et leurs sous-fonctions.

- C'est la fonction qui détermine la forme du système.



Modélisation

Modélisation orienté objet

- La Conception Orientée Objet (COO) est une méthode qui aboutit à des architectures logicielles basées sur les objets du système, plutôt que sur une décomposition fonctionnelle.
- C'est la structure du système lui donne sa forme.
- On peut commencer par les objets du domaine (les éléments fondamentaux) et remonter progressivement vers le système global, ce qui constitue une approche ascendante.

Modélisation

UML

- Au milieu des années 1990, les concepteurs des méthodologies Booch, OOSE et OMT ont décidé d'élaborer un langage de modélisation unifié avec les objectifs suivants :
    - Permettre la modélisation d'un système, depuis les concepts abstraits jusqu'à l'exécutable, en s'appuyant sur les techniques orientées objet ;
    - Réduire la complexité des processus de modélisation ;
    - Offrir une utilisation accessible aussi bien pour les humains que pour les machines, grâce à des représentations graphiques dotées de qualités formelles suffisantes pour une traduction automatique en code source.

Remarque:

- Ces représentations ne possèdent pas des qualités formelles assez robustes pour garantir des propriétés mathématiques rigoureuses.

Modélisation

Les différents types de diagrammes

- En UML, les **diagrammes structurels** et les **diagrammes comportementaux** représentent deux aspects complémentaires d'un système logiciel

  - ❑ **Diagrammes structurels**

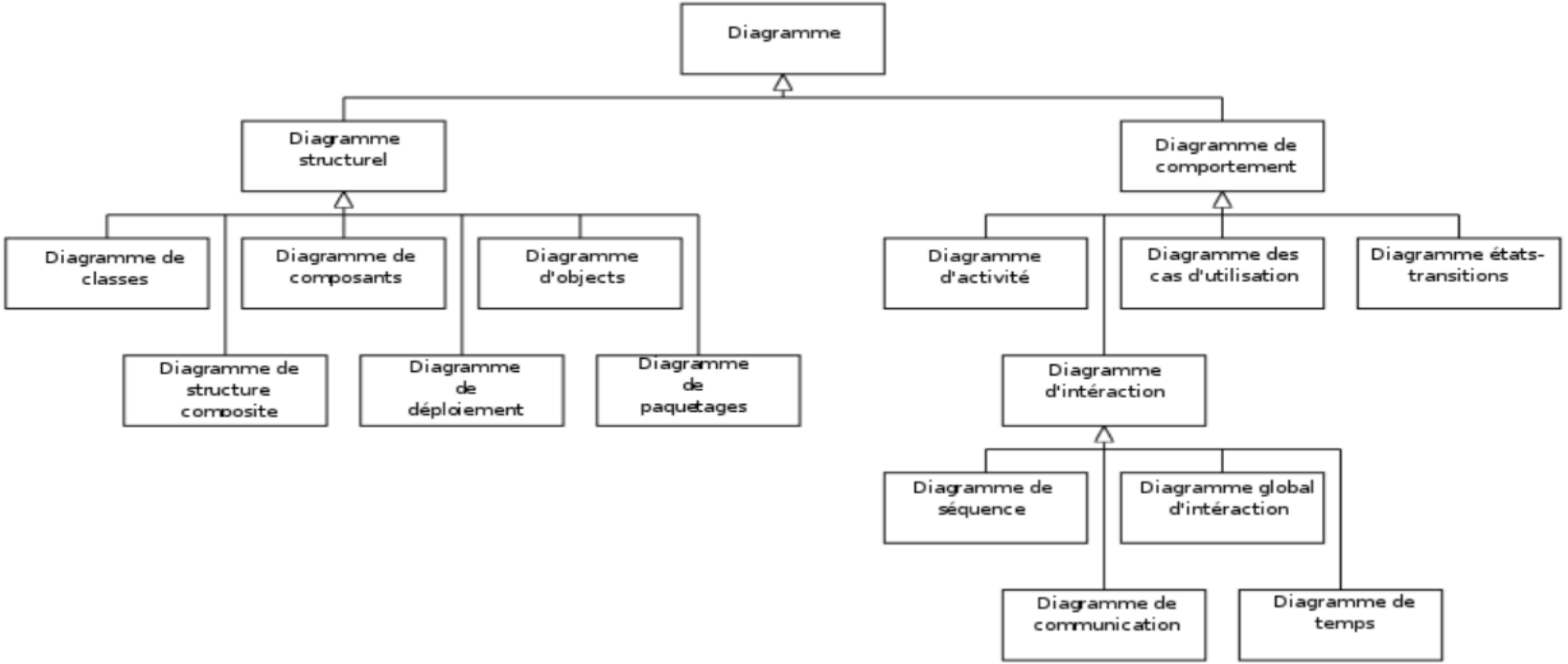
    - Représenter **l'architecture statique** d'un système, c'est-à-dire ses éléments et leurs relations. Ces diagrammes se concentrent sur l'organisation des composants du système à un moment donné, sans montrer leur comportement ou leur dynamique.
    - **Caractéristiques principales :**
      - Décrivent **la structure interne** d'un système (ses objets, classes, composants, ou autres éléments statiques).
      - Ne montrent pas comment ces éléments interagissent ou évoluent dans le temps.

  - ❑ **Diagrammes comportementaux**

    - Représenter les **interactions dynamiques** et le **comportement** du système, c'est-à-dire comment les éléments interagissent entre eux, répondent aux événements ou évoluent dans le temps.
    - **Caractéristiques principales :**
      - Décrivent **les processus, flux d'exécution et interactions** entre les éléments d'un système.
      - Montrent **la dynamique** du système, comme les séquences d'actions ou les réactions à des événements.

Modélisation

Les différents types de diagrammes



Partie 2

Modélisation objet élémentaire avec UML

Le diagramme du cas d’utilisation

Le diagramme du cas d'utilisation

Modélisation des besoins

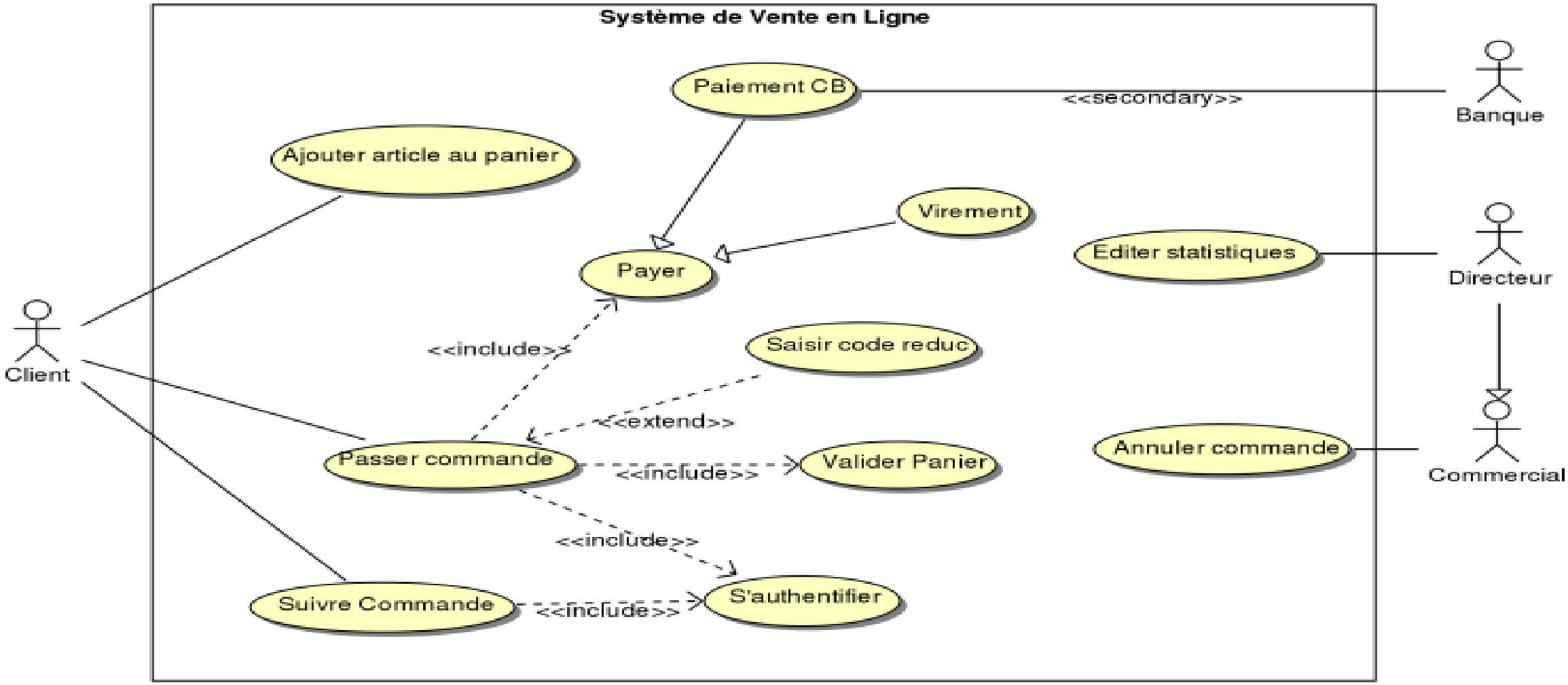
- Avant de développer un système, il est essentiel de déterminer précisément **à quoi il servira**, c'est-à-dire **les besoins qu'il devra satisfaire**.

- La modélisation des besoins permet :
  - **D'identifier et de recenser les fonctionnalités attendues** ;
  - **D'organiser les besoins** afin de mettre en évidence leurs relations, telles que les réutilisations possibles.

- En UML, les besoins sont modélisés à l'aide de **diagrammes de cas d'utilisation**.

Le diagramme du cas d'utilisation

Exemple de diagramme de cas d'utilisation



Le diagramme du cas d’utilisation

Cas d’utilisation

- Un cas d'utilisation est un service fourni à l'utilisateur, impliquant une série d'actions élémentaires.

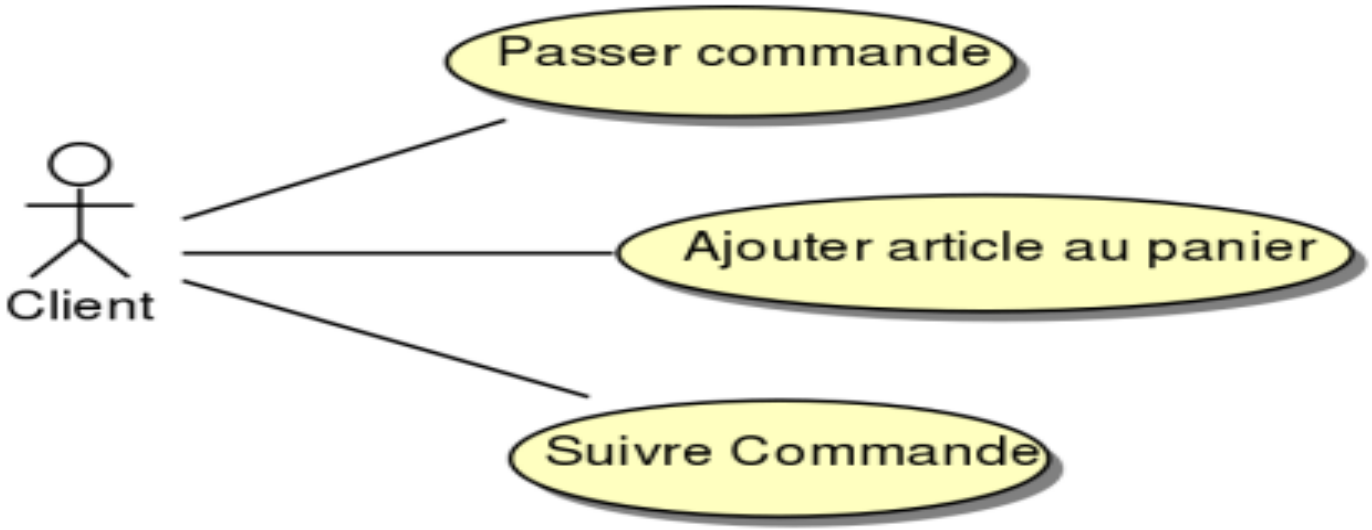
- Un acteur est une entité externe au système modélisé qui interagit directement avec celui-ci.

- Un cas d'utilisation représente un service bout en bout, comprenant un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

Le diagramme du cas d'utilisation

Acteurs

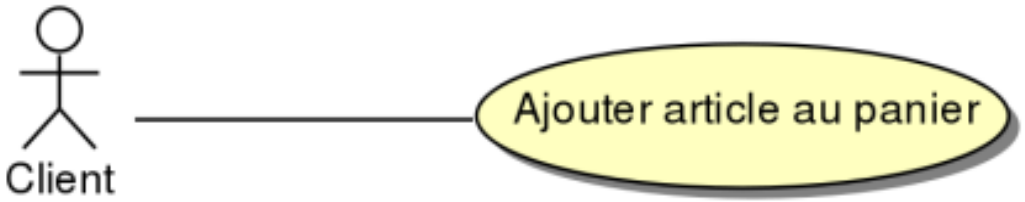
- **Un acteur se présente par un petit bonhomme et un nom(nom du rôle).**



Le diagramme du cas d’utilisation

Relation entre cas d’utilisation en acteurs

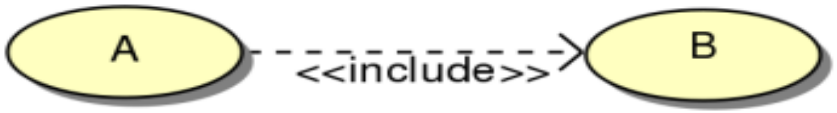
- Les acteurs associés à un cas d'utilisation sont reliés par une association.
- Un acteur peut participer plusieurs fois au même cas d'utilisation.



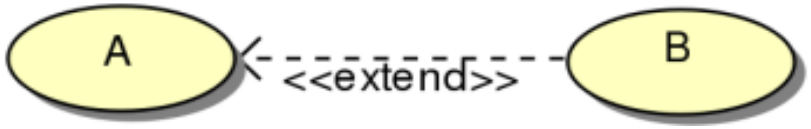
Le diagramme du cas d'utilisation

Relations entre cas d'utilisation

- **Inclusion** : Le cas A inclut le cas B, ce qui signifie que B constitue une partie obligatoire de l'exécution de A.



- **Extension** : Le cas B étend le cas A, ce qui signifie que B est une partie optionnelle dans l'exécution de A.



- **Généralisation** : Le cas A est une généralisation du cas B, ce qui signifie que B est une version spécifique de A.

Le diagramme du cas d'utilisation

Dépendance d'inclusion et d'extension

- Les inclusions et les extensions sont représentées par des relations de dépendance.
  - Lorsqu'un cas B **inclut** un cas A, B dépend de A.
  - Lorsqu'un cas B **étend** un cas A, B dépend également de A.
- La dépendance est notée par une flèche pointillée B⇢A, qui se lit **« B dépend de A »**.
- Si un élément A dépend d'un élément B, toute modification de B peut potentiellement impacter A.
- Les stéréotypes **«include»** et **«extend»** sont des variantes des relations de dépendance.

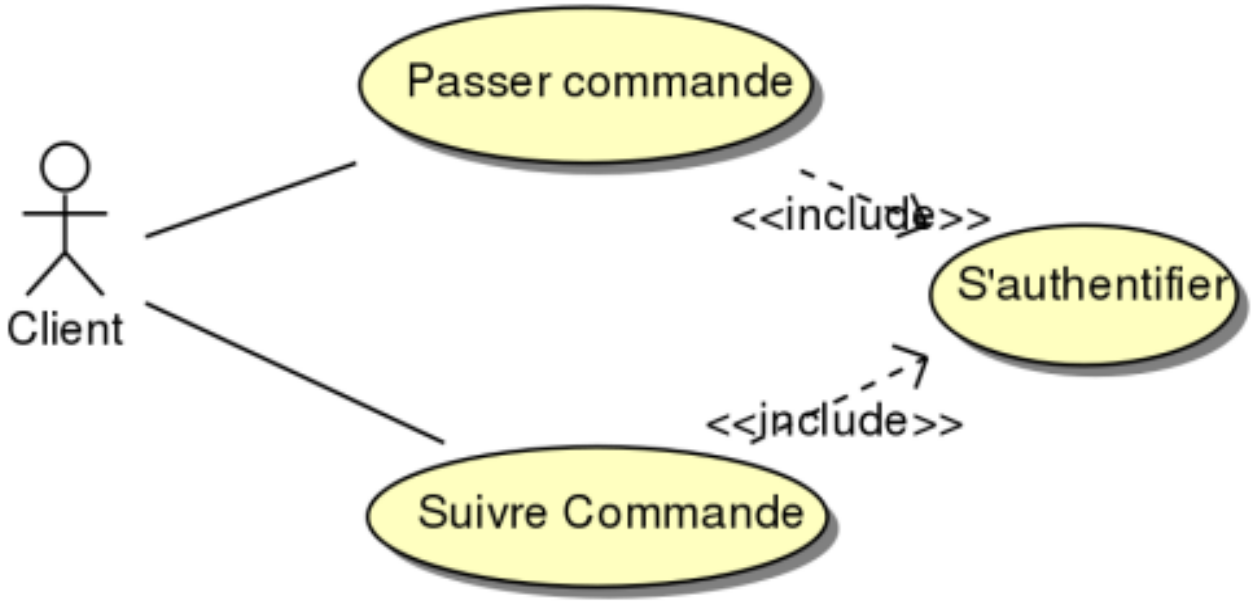
Remarque:

- Le sens des flèches indique la dépendance, et non le sens de la relation d'inclusion ou d'extension.

Le diagramme du cas d'utilisation

Réutilisabilité avec les inclusions et les extensions

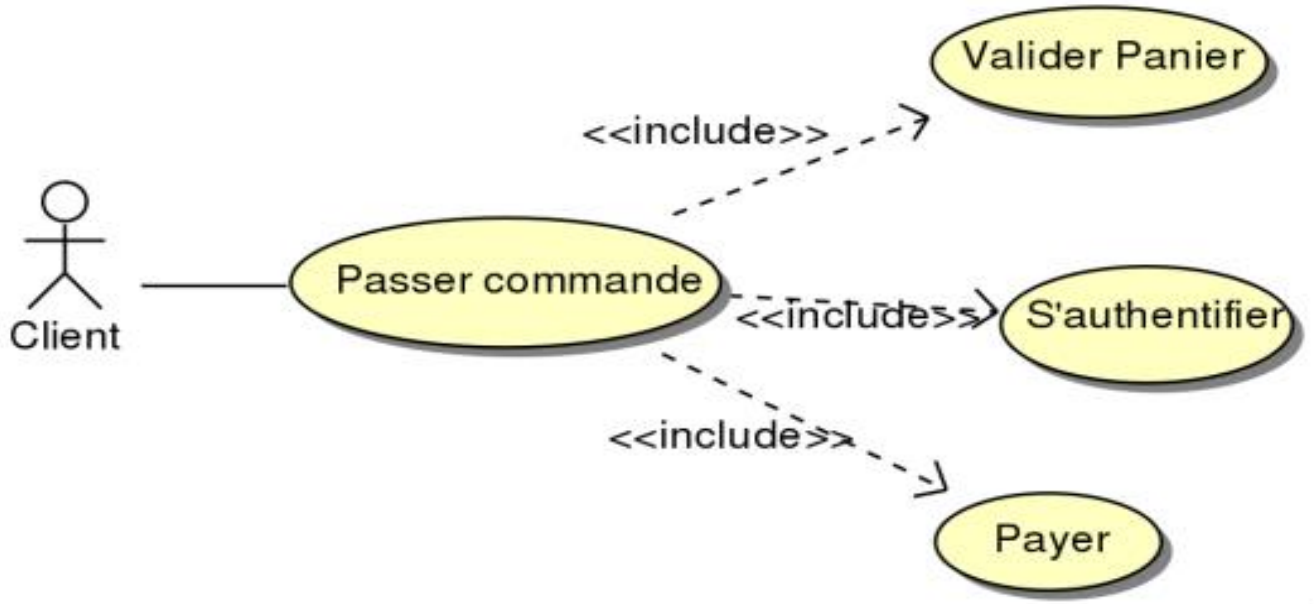
- Les relations entre cas permettent de réutiliser le cas « **s'authentifier** », évitant ainsi de devoir développer plusieurs fois un module d'authentification.



Le diagramme du cas d'utilisation

Décomposition grâce aux inclusions et aux extensions

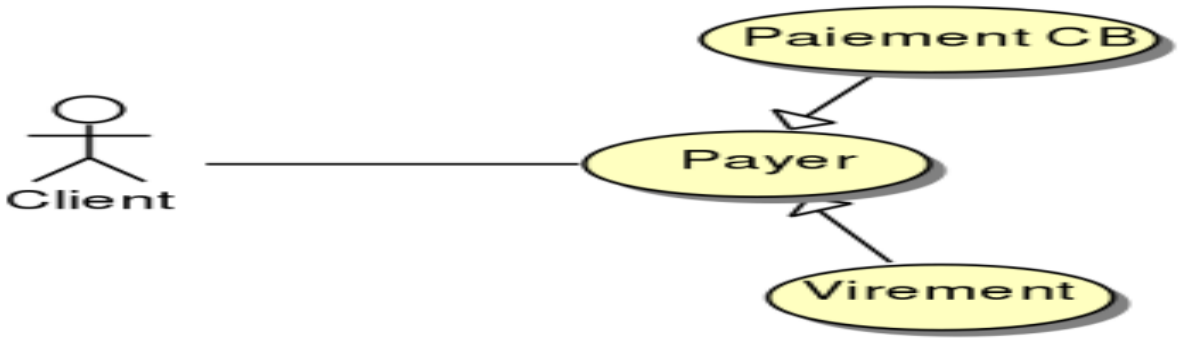
- Lorsqu'un cas est trop complexe, impliquant un trop grand nombre d'actions, il peut être décomposé en cas plus simples.



Le diagramme du cas d'utilisation

Généralisations

- Un virement est un cas spécifique de paiement, ou encore une forme particulière de paiement.

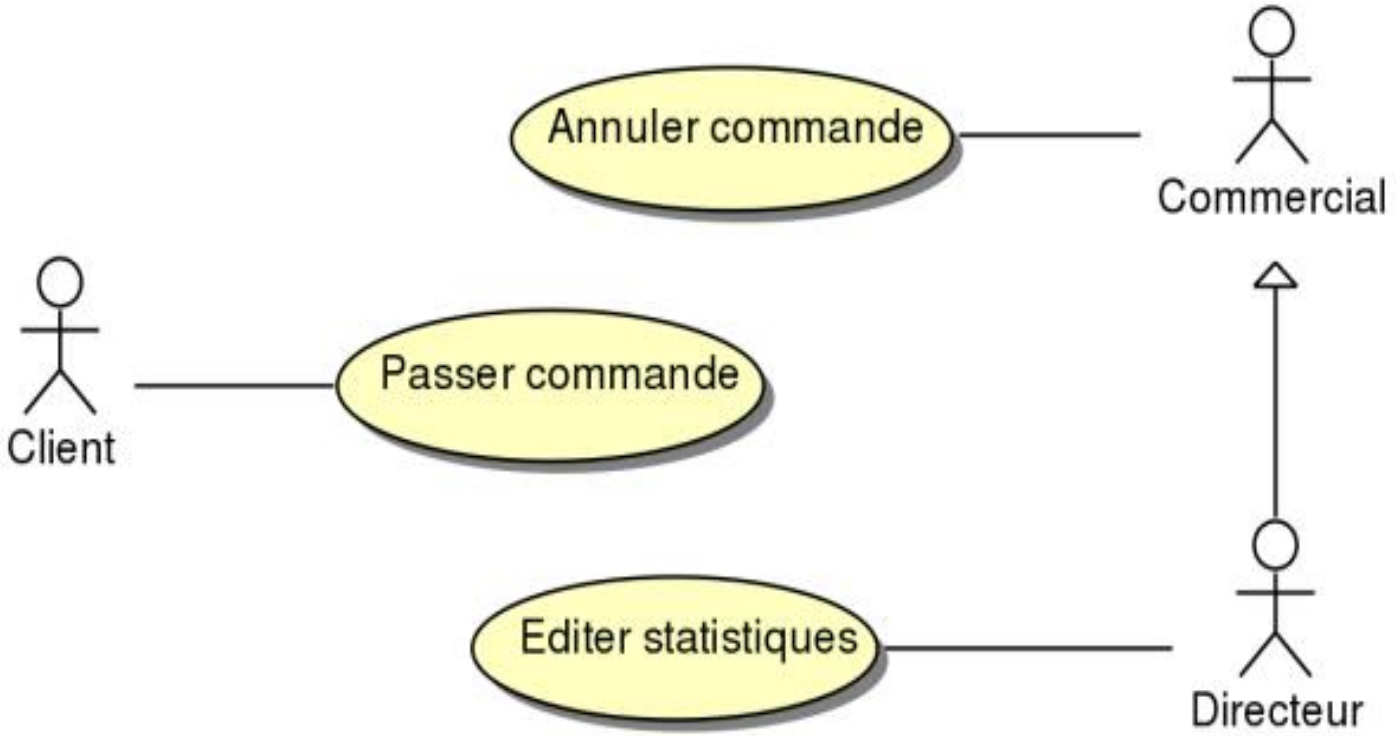


- La flèche pointe vers l'élément général.
- Cette relation de généralisation/spécialisation est courante dans la plupart des diagrammes UML et correspond au concept d'héritage dans les langages de programmation orientés objet.

Le diagramme du cas d'utilisation

Relation entre acteurs

- Une seule relation est possible : la généralisation.



Le diagramme du cas d'utilisation

Identification des acteurs

- Les principaux acteurs sont les utilisateurs du système.

Important : Un acteur représente un rôle, et non une personne physique.

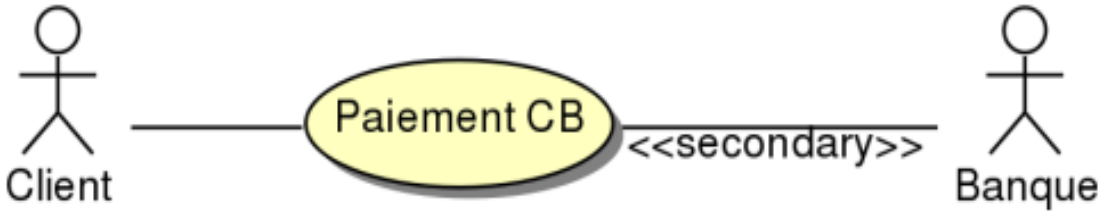
  - Une même personne physique peut être associée à plusieurs acteurs si elle joue plusieurs rôles.
  - Si plusieurs personnes remplissent le même rôle vis-à-vis du système, elles seront représentées par un seul acteur.

- En plus des utilisateurs, les acteurs peuvent inclure :
  - Des périphériques utilisés par le système (comme des imprimantes) ;
  - Des logiciels existants intégrés au projet ;
  - Des systèmes informatiques externes qui interagissent avec le système, etc.
- Pour identifier les acteurs, il est utile de s'appuyer sur les **frontières du système**.

Le diagramme du cas d'utilisation

Acteurs principaux et secondaires

- Les L'acteur est considéré comme principal dans un cas d'utilisation lorsqu'il initie les échanges nécessaires à la réalisation du cas d'utilisation.



- Les acteurs secondaires, quant à eux, sont sollicités par le système, tandis que les acteurs principaux prennent généralement l'initiative des interactions.

- En règle générale, les acteurs secondaires sont d'autres systèmes informatiques avec lesquels le système développé est interconnecté.

Le diagramme du cas d'utilisation

Recenser les cas d'utilisation

- Il n'existe pas de méthode entièrement mécanique et objective pour identifier les cas d'utilisation.
  - Il est nécessaire de se mettre à la place de chaque acteur, de comprendre comment il utilise le système, dans quelles situations il l'interagit et quelles fonctionnalités il doit pouvoir exploiter.
  - Il est important d'éviter les répétitions et de limiter le nombre de cas d'utilisation en choisissant le bon niveau d'abstraction (par exemple, éviter de réduire un cas à une seule action).
  - Il convient également de ne pas inclure les détails techniques des cas, mais de se concentrer sur les grandes fonctionnalités du système.
- **Trouver le niveau de détail approprié pour les cas d'utilisation est un défi qui demande de l'expérience.**

Le diagramme du cas d’utilisation

Description des cas d’utilisation

- Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.
- Un simple nom est tout à fait insuffisant pour décrire un cas d'utilisation.
- *Chaque cas d'utilisation doit être documenté pour qu'il n'y ait aucune ambiguïté concernant son déroulement et ce qu'il recouvre précisément .*

Le diagramme du cas d'utilisation

Description textuelle: identification

- Identification :

Nom du cas : Effectuer un paiement par carte bancaire
Objectif : Décrire les étapes permettant au client de réaliser un paiement par carte bancaire
Acteurs : Client, Banque (secondaire)
Date : 25/12
Responsable : Mohamed
Version : 1.0

Le diagramme du cas d'utilisation

Description textuelle: séquencements

❑ Séquencements :

- **Le cas d'utilisation débute lorsque le client choisit de payer par carte bancaire.**

- **Pré-conditions**
  - **Le client a validé sa commande.**
- **Enchaînement nominal**
  1. **Le client saisit les informations de sa carte bancaire.**
  2. **Le système vérifie que le numéro de la carte est valide.**
  3. **Le système valide la carte auprès du système bancaire.**
  4. **Le système demande au système bancaire de procéder au débit du client.**
  5. **Le système informe le client que la transaction a été réalisée avec succès.**
- **Enchaînements alternatifs**
  1. **À l'étape (2) : Si le numéro de carte est incorrect, le client est informé de l'erreur et invité à saisir à nouveau les informations.**
  2. **À l'étape (3) : Si les informations sont erronées, le système demande à nouveau les données au client.**
- **Post-conditions**
  - **La commande est validée.**
  - **Le compte de l'entreprise est crédité.**

Le diagramme du cas d'utilisation

Description textuelle: séquencements

❑ **Rubriques optionnelles**

- **Contraintes non fonctionnelles :**
  - **Fiabilité :** Les accès au système doivent être sécurisés pour garantir la protection des transactions.
  - **Confidentialité :** Les données personnelles du client doivent rester strictement confidentielles et ne pas être divulguées.
- **Contraintes liées à l'interface utilisateur :**
  - Toujours demander une validation explicite des opérations bancaires par l'utilisateur.

Le diagramme du cas d'utilisation

StarUML

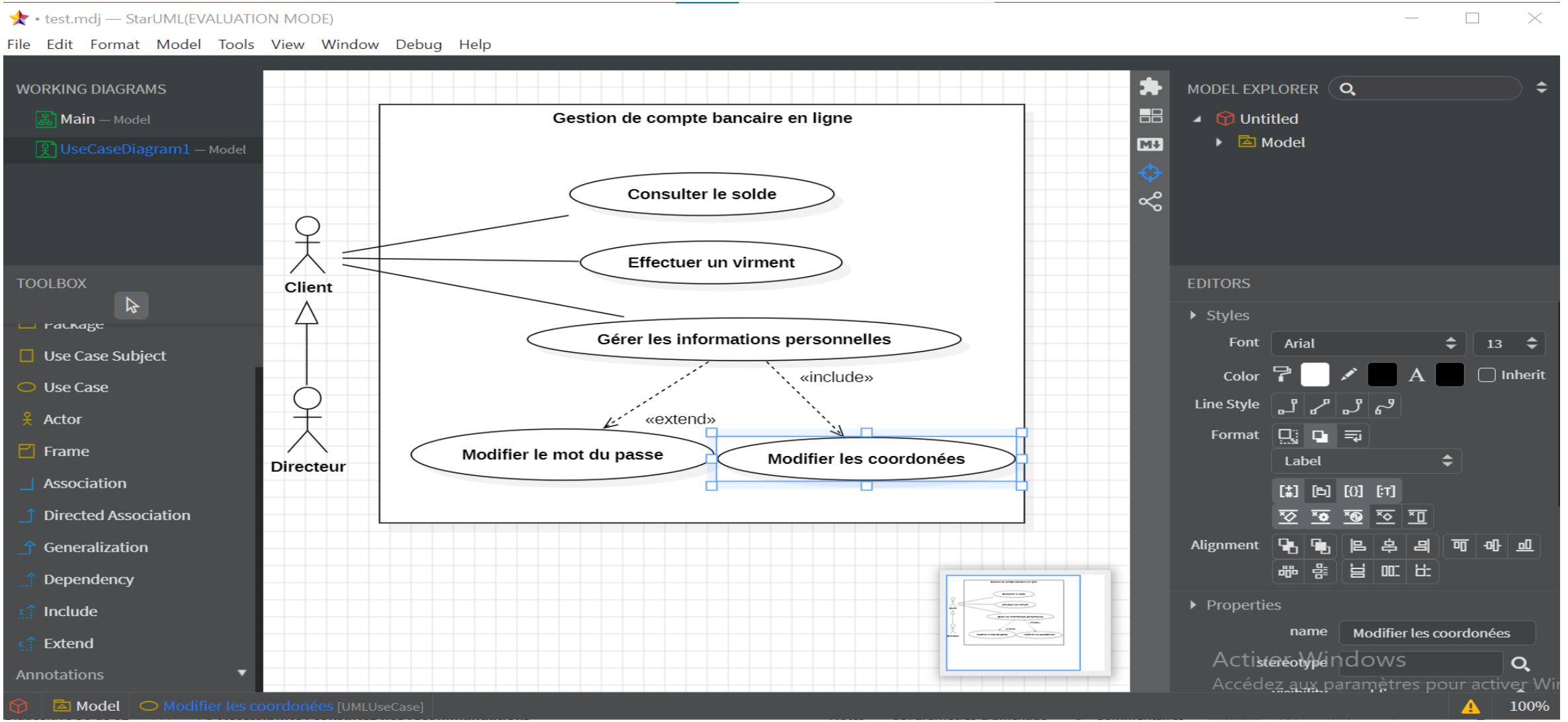


Diagramme de classes

Diagramme de classes

Objectif

- Les diagrammes de cas d'utilisation décrivent **le rôle et les objectifs** du système.
- Le système est constitué d'objets qui interagissent entre eux et avec les acteurs pour accomplir les différents cas d'utilisation.
- Quant aux **diagrammes de classes**, ils spécifient la structure interne du système et les relations entre les objets qui le composent.

Diagramme de classes

Exemple de diagramme de classes

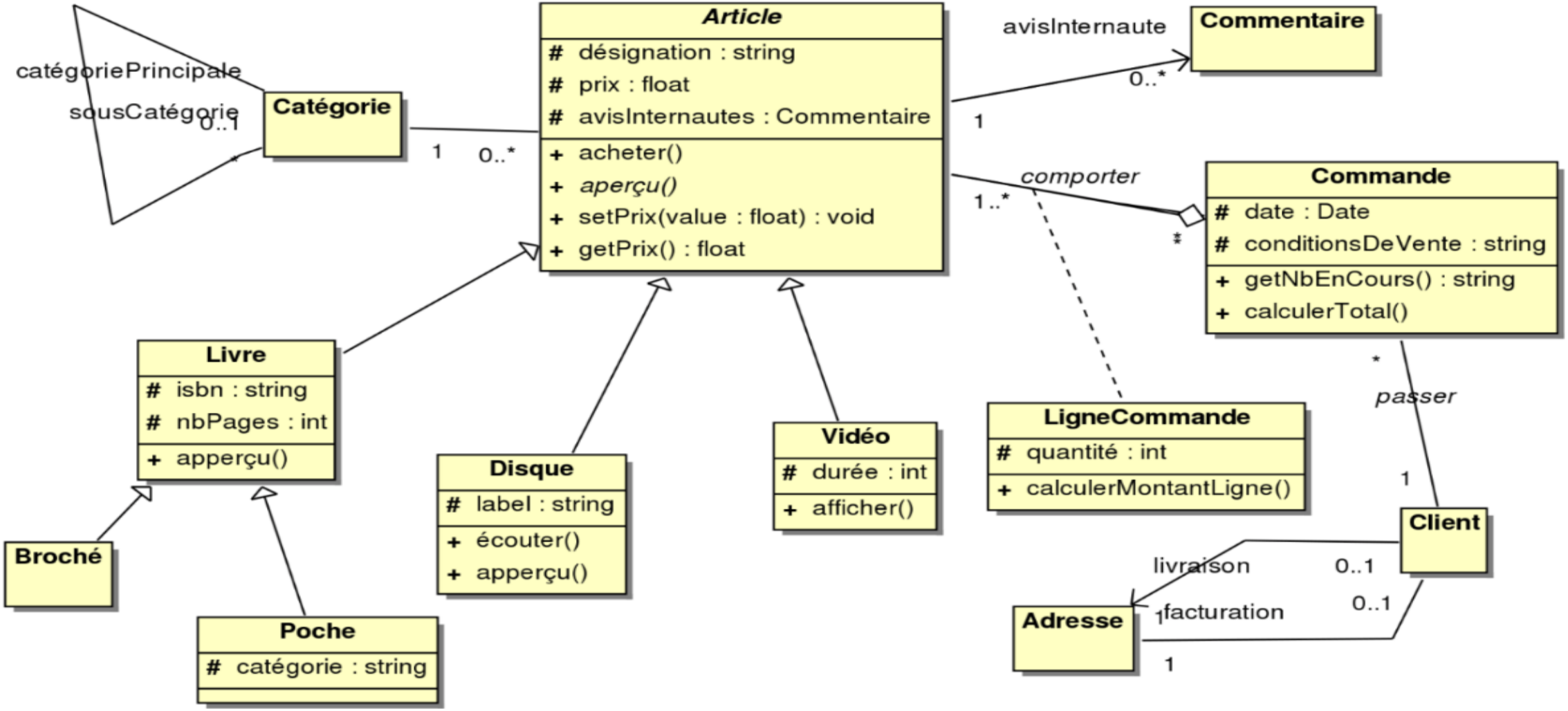


Diagramme de classes

Concepts et instances

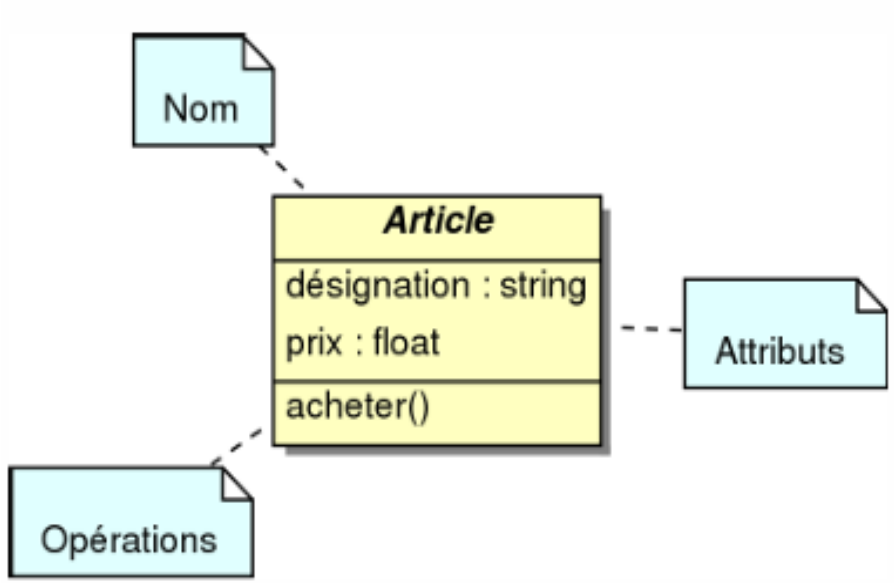
- Une instance représente la matérialisation d'un concept abstrait.
  - Concept : Stylo
  - Instance : Le stylo que vous utilisez en ce moment est une instance du concept de stylo : il possède des caractéristiques spécifiques comme sa forme, sa couleur ou son niveau d'usure.
- Un objet est une instance d'une classe.
  - Classe : Voiture
  - Objets : Ford, Mercedes

    Une classe décrit un type d'objets concrets.
  - Elle définit comment tous les objets de ce type sont caractérisés (par exemple, leur couleur, leur modèle, etc.).
- Un lien représente une instance d'une association.
  - Association : Le concept d'avis d'internaute, qui relie un commentaire à un article.
  - Lien : Une instance de cette association, comme [Jean avec son avis négatif] ou [Paul avec son avis positif].

Diagramme de classes

Classes et objets

- Une classe définit un ensemble d'objets partageant une même sémantique, des attributs, des méthodes, et des relations communes.
- Elle spécifie les caractéristiques qui définissent les objets d'un même type.
- Une classe se compose d'un **nom**, d'**attributs**, et d'**opérations**.
- En fonction de l'état d'avancement de la modélisation, certaines de ces informations peuvent ne pas être encore définies.



- Des éléments supplémentaires, tels que les responsabilités ou les exceptions, peuvent également être ajoutés si nécessaire.

Diagramme de classes

Propriété : attributs et opérations

- Les attributs et les opérations représentent les propriétés d'une classe et sont généralement nommés avec une lettre initiale en minuscule.
  - Un **attribut** correspond à une donnée spécifique de la classe.
    - Le modèle peut préciser le type des attributs, leurs valeurs initiales, ainsi que les modificateurs d'accès.
    - Lorsqu'une classe est instanciée, les attributs prennent des valeurs : ils fonctionnent comme des variables associées aux objets.
  - Une **opération** désigne un service fourni par la classe, c'est-à-dire une action ou un traitement que les objets de cette classe peuvent effectuer.

Diagramme de classes

- **Comportement des attributs**

- Un attribut peut être initialisé, et sa visibilité est définie au moment de sa déclaration.

Syntaxe

modifAcces nomAtt : nomClasse [multi] = valeurInit

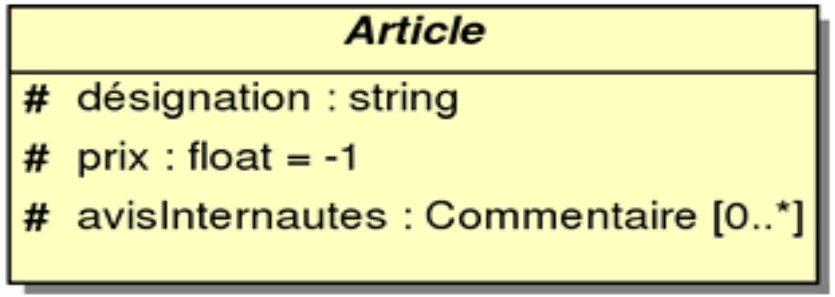


Diagramme de classes

- **Comportement des opérations**

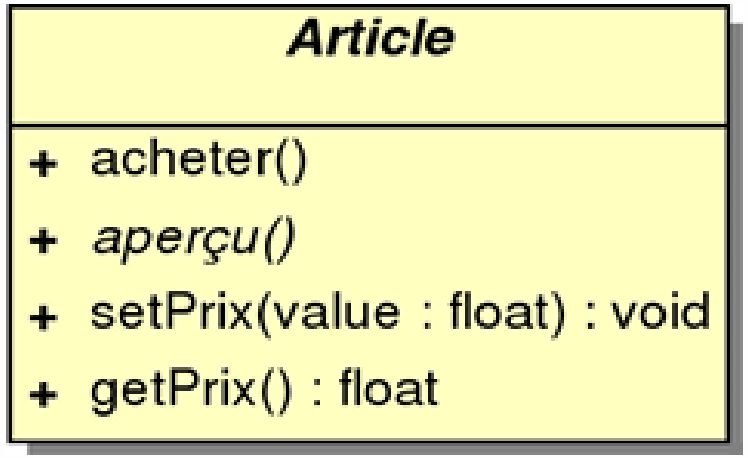
- Une opération est définie par son nom, les types de ses paramètres, et le type de sa valeur de retour.

Syntaxe pour déclarer une opération :

modifAcces nomOperation(parametres) : ClasseRetour

Syntaxe pour la liste des paramètres :

nomClasse1 nomParam1, ..., nomClasseN nomParamN



- Une opération spécifie une méthode par sa signature, sans

tenir compte de son implémentation.

Diagramme de classes

- **Relation entre classes**

- Une relation d'héritage représente un lien de généralisation ou de spécialisation, favorisant l'abstraction.
- Une dépendance est un lien unidirectionnel qui exprime une dépendance sémantique entre les éléments du modèle (représentée par une flèche ouverte en pointillés).
- Une association symbolise une relation sémantique entre les objets d'une classe.
- Une relation d'agrégation illustre un lien de contenance ou de composition.

Diagramme de classes

- **Association**

- Une association est un lien structurel entre des objets.
- Elle sert généralement à représenter les relations possibles entre des objets de classes spécifiques.
- Elle est symbolisée par une ligne reliant les classes concernées.

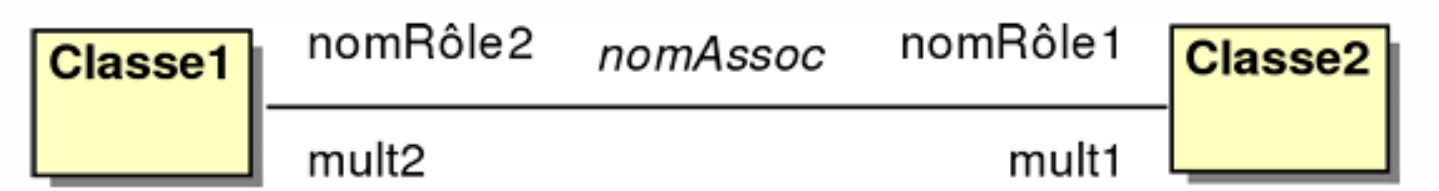


Diagramme de classes

• Multiplicités des associations

- La multiplicité permet de spécifier le nombre d'objets impliqués dans chaque instance d'une association.

Exemple :

- Un article appartient à une seule catégorie (1).
- Une catégorie peut inclure au moins un article, sans limite maximale (*).

Syntaxe :

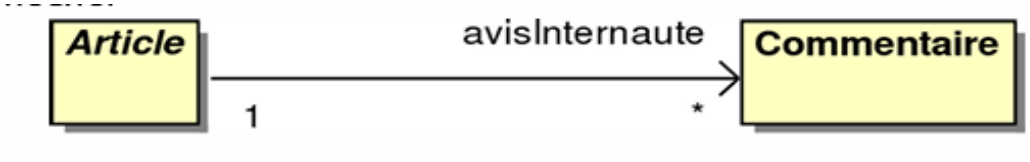
MultMin..MultMax

- Le symbole * pour MultMax indique un nombre illimité.
- n..n peut être abrégé en n.
- 0..* est également simplifié en *.

Diagramme de classes

• Navigabilité d'une association

- La navigabilité indique dans quel(s) sens il est possible de parcourir une association lors de l'exécution.
- Pour limiter la navigabilité à un seul sens, une flèche est utilisée.



Exemple : En connaissant un article, on peut accéder à ses commentaires, mais l'inverse n'est pas possible.

- Les associations navigables dans un seul sens peuvent également être représentées par des attributs.

Exemple : Ajouter un attribut « avisInternaute » de type « Commentaire » dans la classe Article, à la place de l'association.

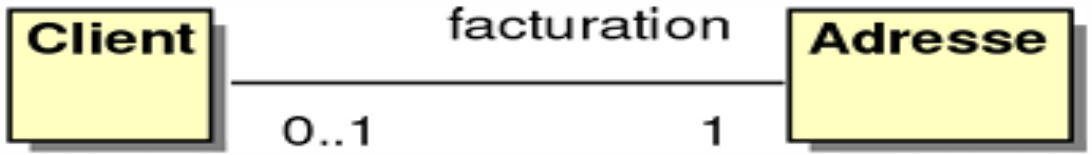
Diagramme de classes

- **Association unidirectionnelle de 1 vers 1**

- Une flèche pointe de **Client** vers **Adresse**, ce qui indique que la relation est **navigable dans un seul sens**.
- Cela signifie que, connaissant un client, on peut retrouver son adresse de livraison, mais pas l'inverse

Diagramme de classes

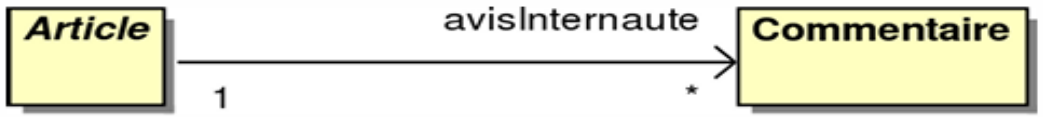
- **Association bidirectionnelle de 1 vers 1**



- Il n'y a **aucune flèche** indiquée sur la relation.
- Cela suggère que la relation est **bidirectionnelle**, ce qui signifie qu'il est possible de naviguer :
  - De **Client** vers **Adresse** pour retrouver l'adresse de facturation.
  - De **Adresse** vers **Client** pour retrouver le client correspondant.

Diagramme de classes

- **Association unidirectionnelle de 1 vers ***



- Une flèche pointe de **Article** vers **Commentaire**, ce qui indique que la relation est **navigable dans un seul sens**.
- Multiplicité

  Côté Article :

  - Multiplicité 1.
  - Cela signifie qu'un commentaire est toujours lié à **un seul article**.

  Côté Commentaire :

  - Multiplicité *.
  - Cela indique qu'un article peut avoir **plusieurs commentaires.**

Diagramme de classes

- **Association unidirectionnelle de * vers ***

- Une association bidirectionnelle de * vers * relie deux classes où chaque instance de l'une peut être associée à plusieurs instances de l'autre, et vice versa.
- Elle est représentée par une ligne entre les classes Article et Commande, sans flèche, indiquant que la navigation est possible dans les deux sens.

Diagramme de classes

- **Associations réflexives**

- L'association la plus courante est l'association binaire, qui relie deux classes.
- Parfois, les deux extrémités de l'association pointent vers la même classe.
- Dans ce cas, on parle d'une association réflexive.

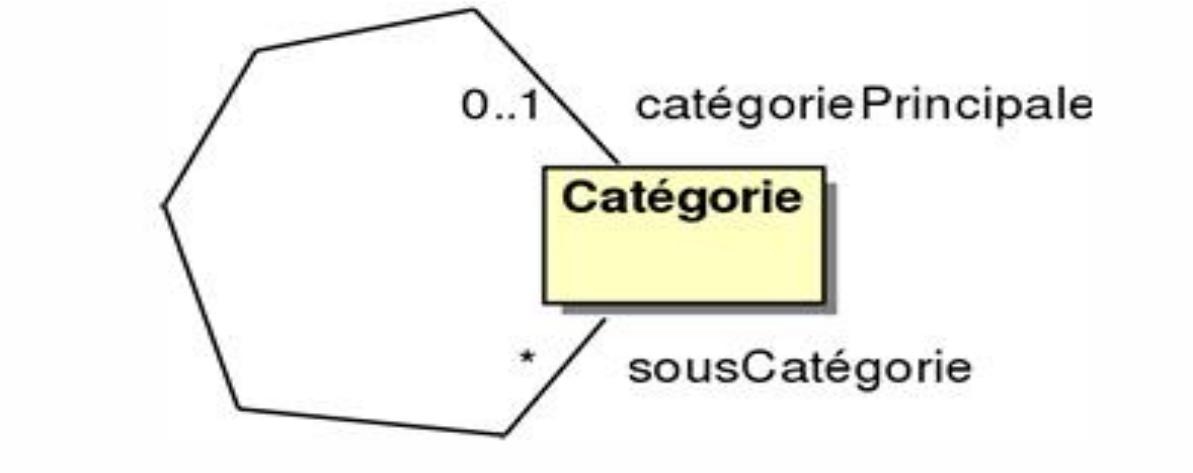


Diagramme de classes

- **Classe-association**

- Une classe-association est une association qui possède ses propres attributs ou opérations.

- Elle relie deux classes tout en agissant comme une classe à part entière, avec ses propres propriétés.

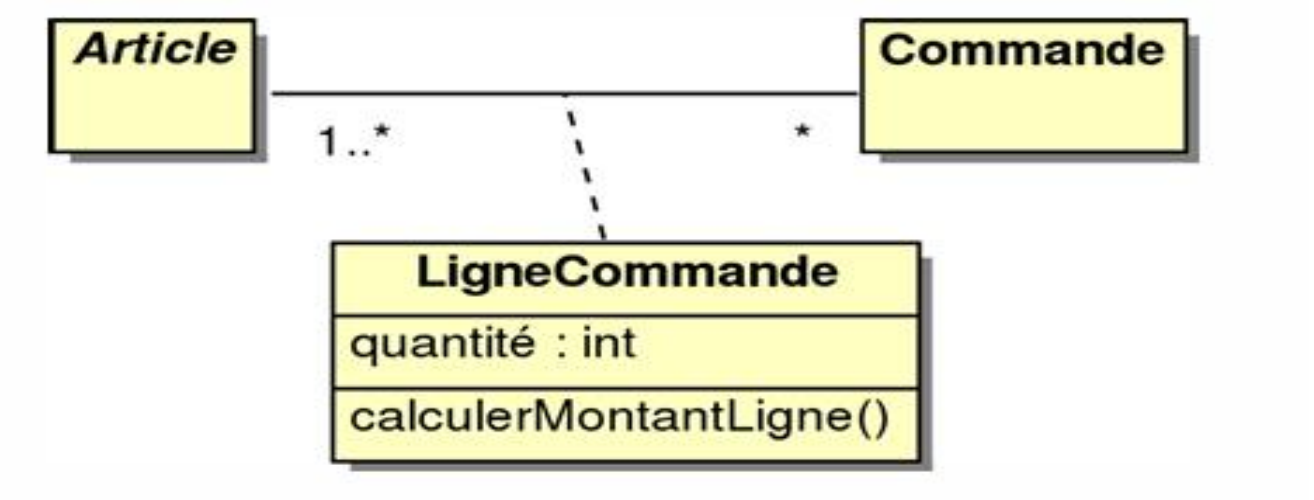


Diagramme de classes

• Associations n-aires

- Une **association n-aire** relie plus de deux classes.
- Elle est représentée par un losange central qui peut inclure une classe-association.
- La multiplicité de chaque classe s'applique à une instance du losange.
- Ces associations sont rares et s'utilisent principalement lorsque toutes les multiplicités sont *.
- Dans la plupart des cas, il est préférable d'utiliser des classes-association ou plusieurs associations binaires.

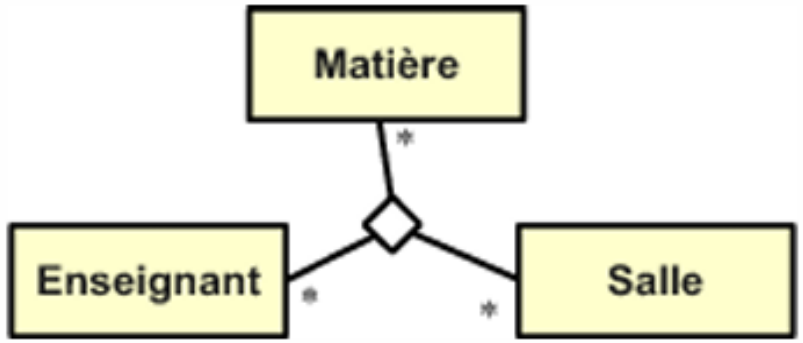


Diagramme de classes

• Association de type agrégation

- L'agrégation est un type d'association. Elle montre qu'un élément fait partie d'un ensemble.
- Dans un diagramme, l'agrégation est représentée par un losange vide près de l'ensemble (l'agrégat).
- L'agrégation montre la relation entre un tout (l'agrégat) et ses parties (les agrégés).

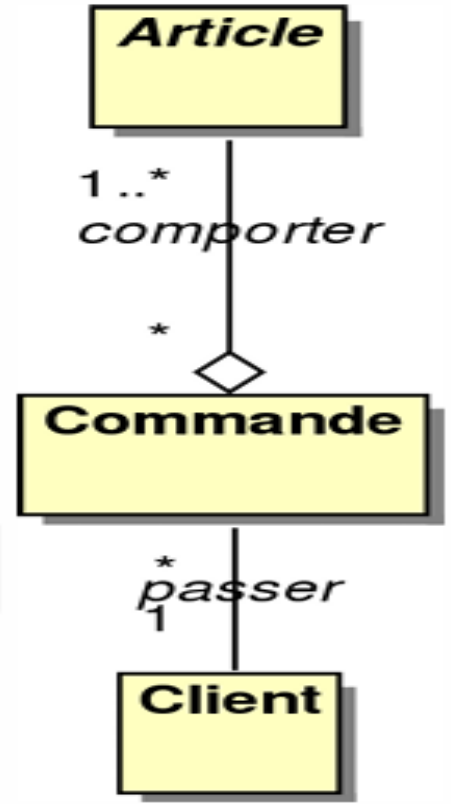


Diagramme de classes

- **Association de type composition**

- La composition représente une relation où un objet contient d'autres objets.
- Elle est symbolisée par un losange plein.
- Si l'objet principal est détruit ou copié, ses composants le sont aussi.
- Une partie ne peut appartenir qu'à un seul objet composite.

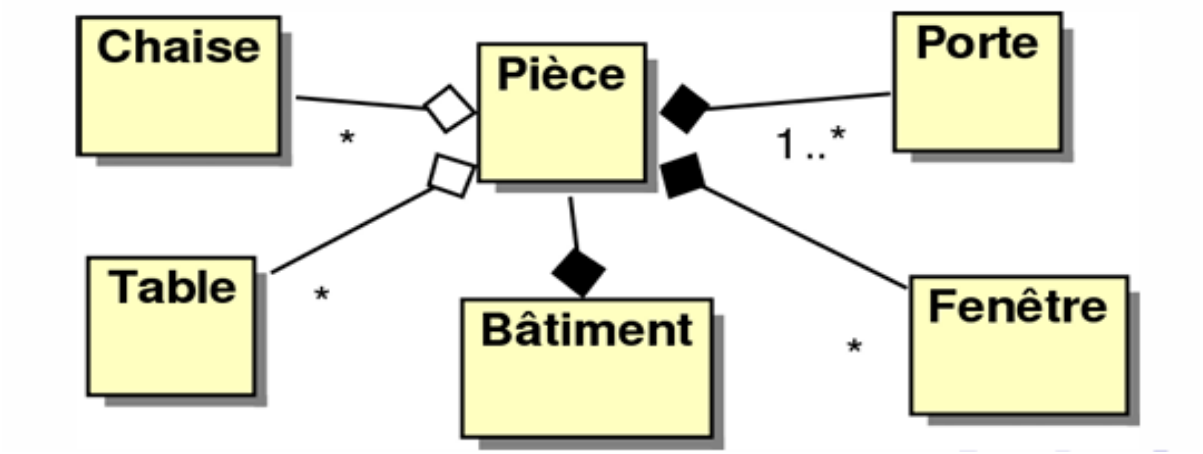


Diagramme de classes

• Composition et agrégation

- Quand il y a une relation entre un tout et ses parties, il s'agit d'agrégation ou de composition.
- La composition est une forme forte d'agrégation.
- Pour choisir entre composition et agrégation, posez-vous ces questions :
- La destruction de l'ensemble doit-elle entraîner celle de ses parties ? C'est le cas si les parties ne peuvent exister sans l'ensemble.
- Copier l'ensemble implique-t-il de copier aussi ses parties, ou peut-on réutiliser les parties dans plusieurs ensembles ?
- Si la réponse est oui à ces deux questions, il faut utiliser la composition.

Diagramme de classes

• Relation d'héritage

- L'héritage est une relation de spécialisation/généralisation.
- Les éléments spécialisés héritent des caractéristiques et comportements des éléments généraux (attributs et opérations).

Exemple : Un livre hérite d'un article, donc il a un prix,

une désignation et une opération acheter(),

sans qu'il soit nécessaire de les définir à nouveau.

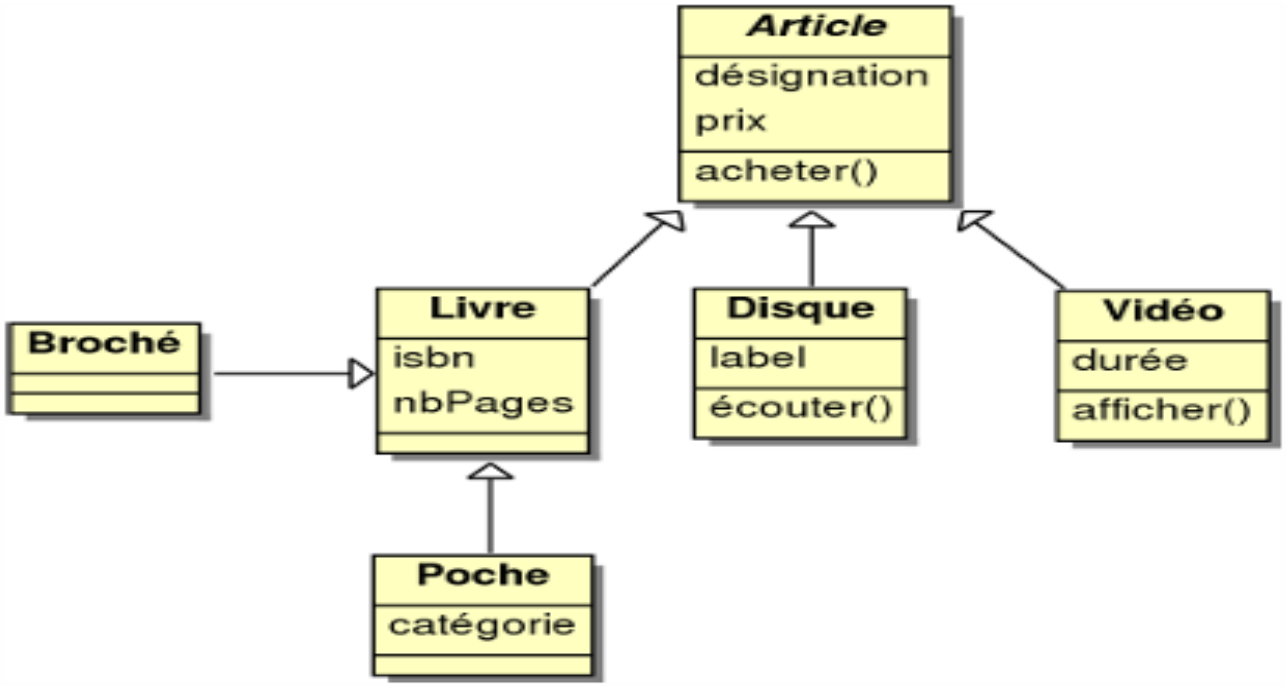


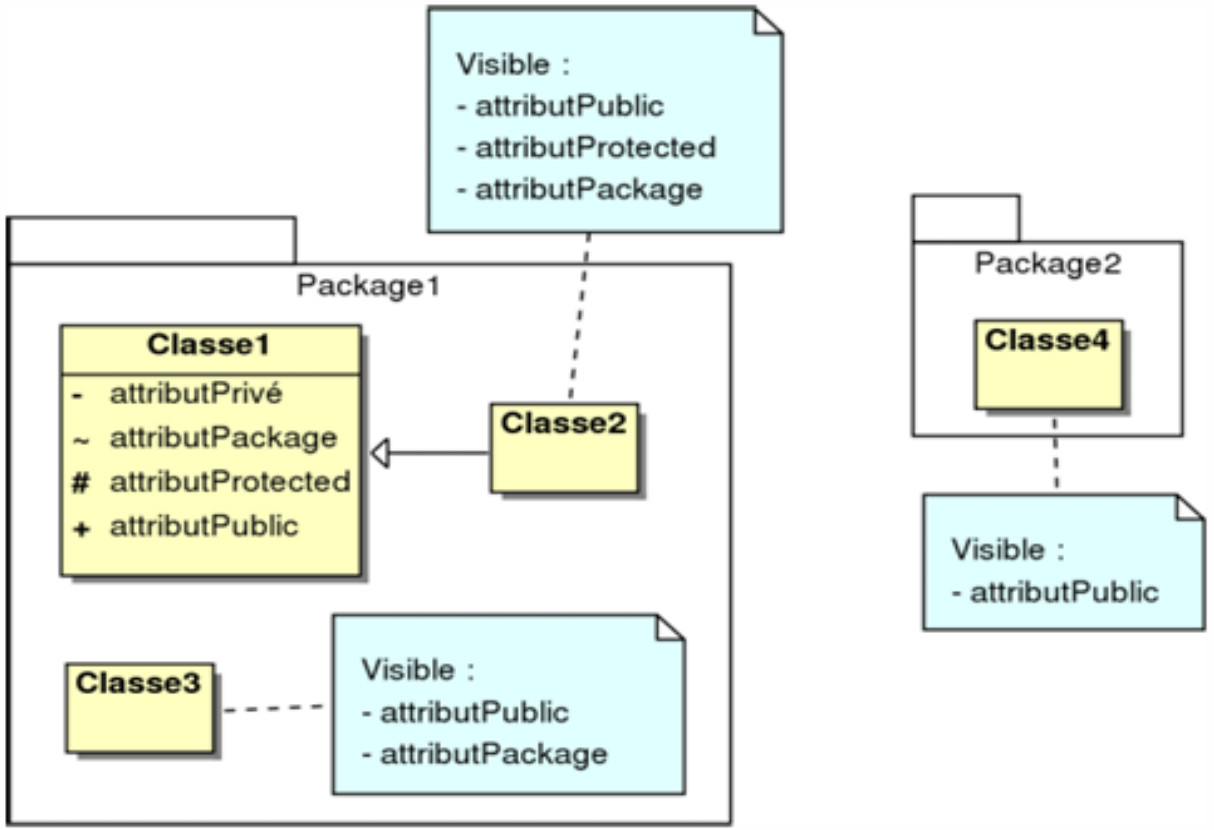
Diagramme de classes

- **Encapsulation**

- L'encapsulation est un principe qui protège le cœur d'un système contre les accès non autorisés venant de l'extérieur.
- En UML, on utilise des modificateurs d'accès pour les attributs ou les classes :
  - Public « + » : visible partout.
  - Protected « # »: visible dans la classe et ses sous-classes.
  - Private « - » : visible uniquement dans la classe.
  - Package « ~ » : visible uniquement dans le paquetage.
- Il n'y a pas de visibilité par défaut.
- Les **packages** regroupent des éléments de modélisation comme des classes, des diagrammes de cas d'utilisation ou d'autres packages. Les éléments sont organisés en packages et sous-packages.

Diagramme de classes

- **Exemple d’ encapsulation**



Les modificateurs d'accès s'appliquent aussi aux opérations

Diagramme de classes

• Relation d’héritage et propriété

- La classe enfant hérite des propriétés (attributs et méthodes) de ses classes parents.
  - La classe enfant est la classe spécialisée (ici, **Livre**).
  - La classe parent est la classe générale (ici, **Article**).
- Cependant, la classe enfant n'a pas accès aux propriétés privées de la classe parent.

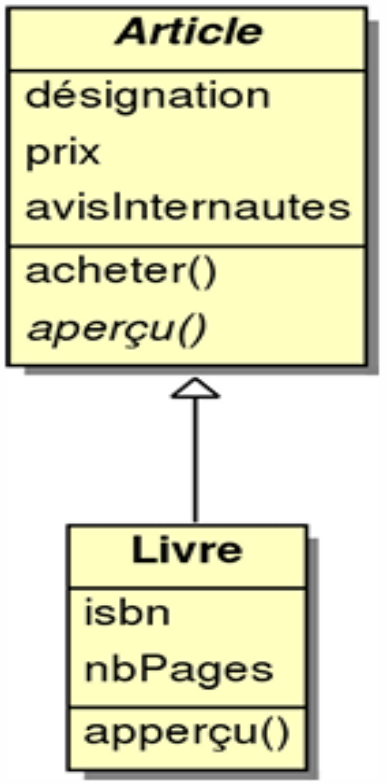


Diagramme de classes

• Classes abstraites

- Une méthode est abstraite lorsque son entête est définie, mais pas son implémentation.
- Les classes enfants doivent définir les méthodes abstraites.
- Une classe est abstraite si elle contient au moins une méthode abstraite ou si elle hérite d'une méthode abstraite non encore implémentée.

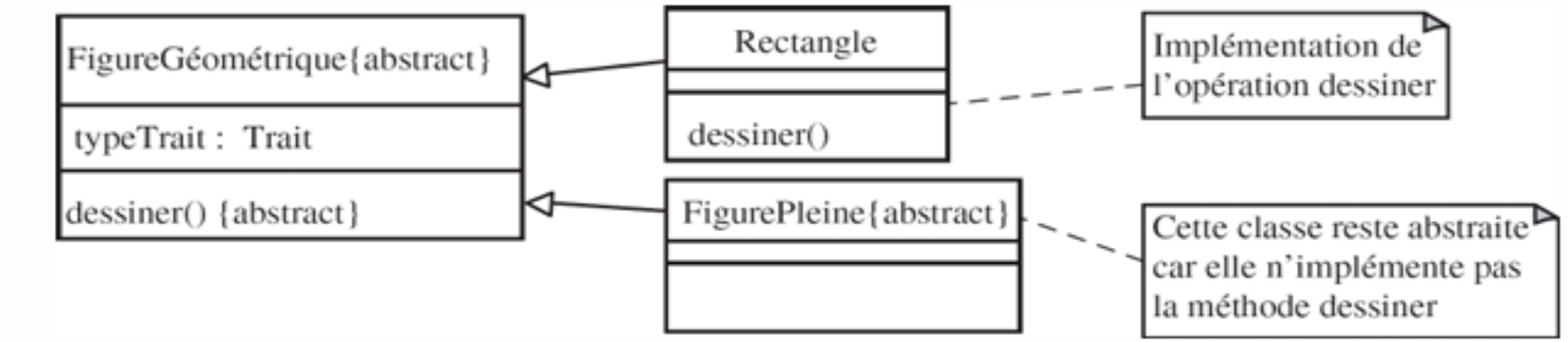


Diagramme de classes

- **Interface**

- Une interface regroupe un ensemble d'opérations qui assurent un service cohérent pour une ou plusieurs classes.
- Elle est définie comme une classe, avec les mêmes sections, mais le stéréotype « interface » est ajouté avant le nom.
- Une classe qui implémente une interface est liée à celle-ci par une relation de réalisation.
- Les classes qui implémentent une interface doivent inclure toutes les opérations définies dans cette

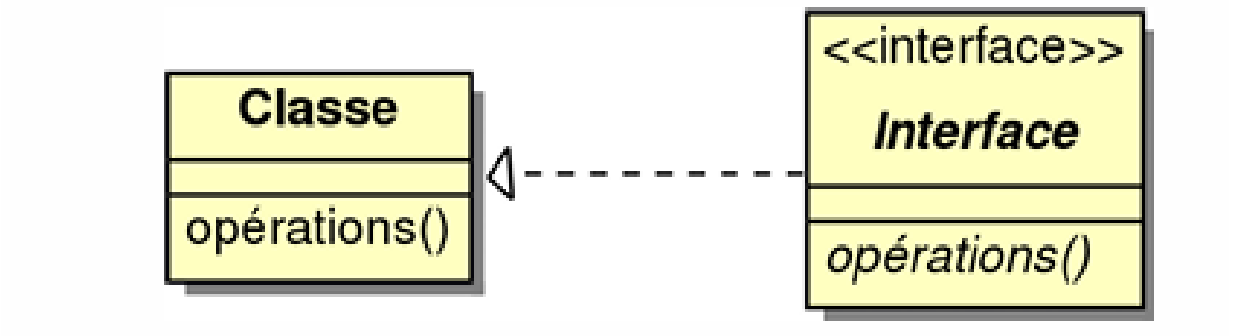
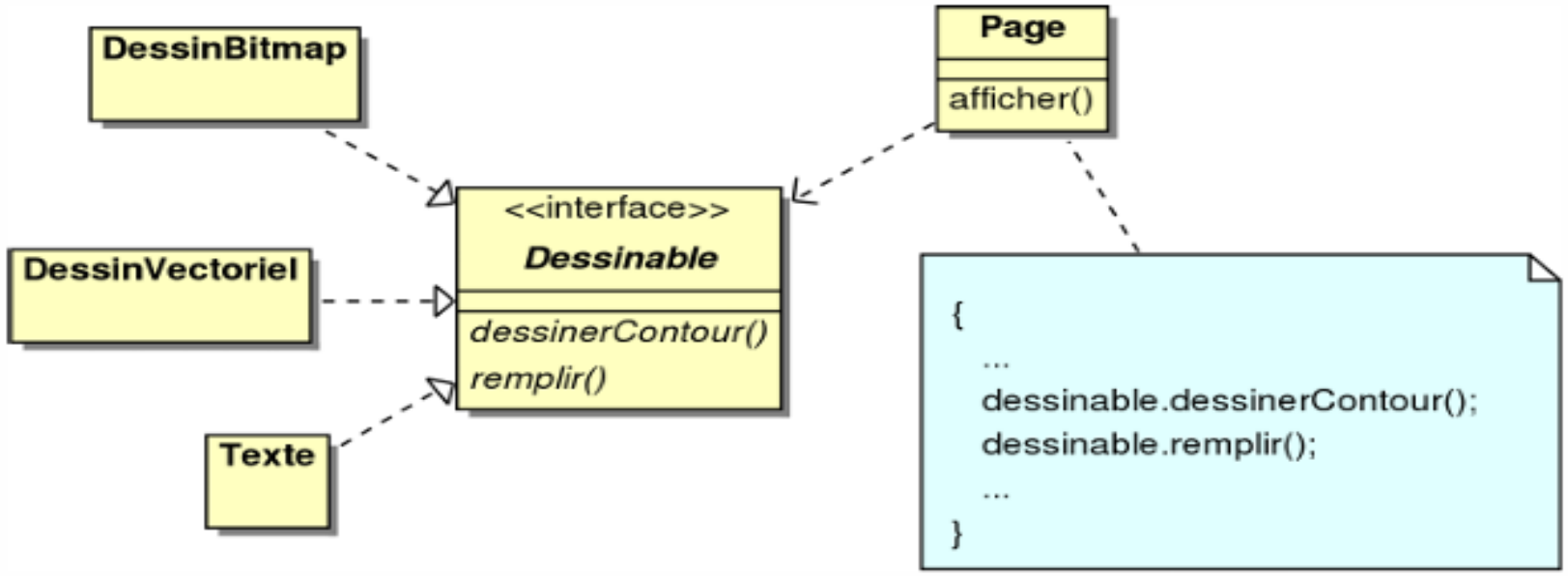


Diagramme de classes

- **Exemple d'interface**



Remarque :

- Lorsqu'une classe utilise une interface pour réaliser ses opérations, elle est appelée classe cliente de l'interface.

Diagramme de classes

- **Eléments dérivés**

- Les attributs dérivés sont calculés à partir d'autres attributs ou de formules.
- Ils sont représentés par un slash « / » avant leur nom.
- Lors de la conception, un attribut dérivé peut servir de marqueur en attendant de définir les règles à appliquer.
- Une association dérivée dépend ou peut être déduite d'autres associations.
- Elle utilise aussi le symbole « /».

Diagramme de classes

• Attributs de classe

- Par défaut, les valeurs des attributs d'une classe sont différentes pour chaque objet.
- Parfois, il est nécessaire de définir un attribut de classe qui a une valeur unique, partagée par toutes les instances.
- Un attribut de classe est représenté par un soulignement.

Diagramme de classes

- **Opérations de classe**

- Comme les attributs de classe, une opération de classe est une propriété de la classe elle-même, et non de ses instances.

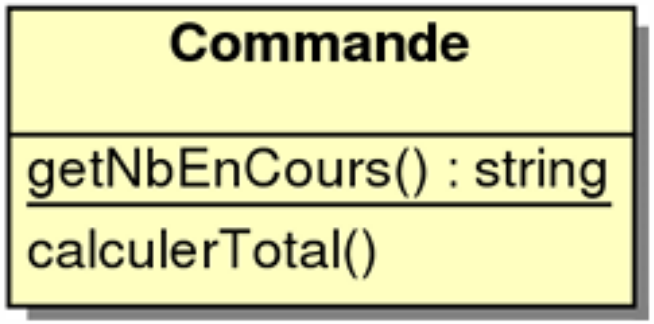


Diagramme de classes

- **Classe paramétré**

- Pour définir une classe générique et paramétrable selon des valeurs ou des types :
  - On définit la classe avec des éléments spécifiés dans un rectangle en pointillés.
  - On utilise une dépendance stéréotypée « bind » pour créer des classes en fonction de la classe paramétrée.

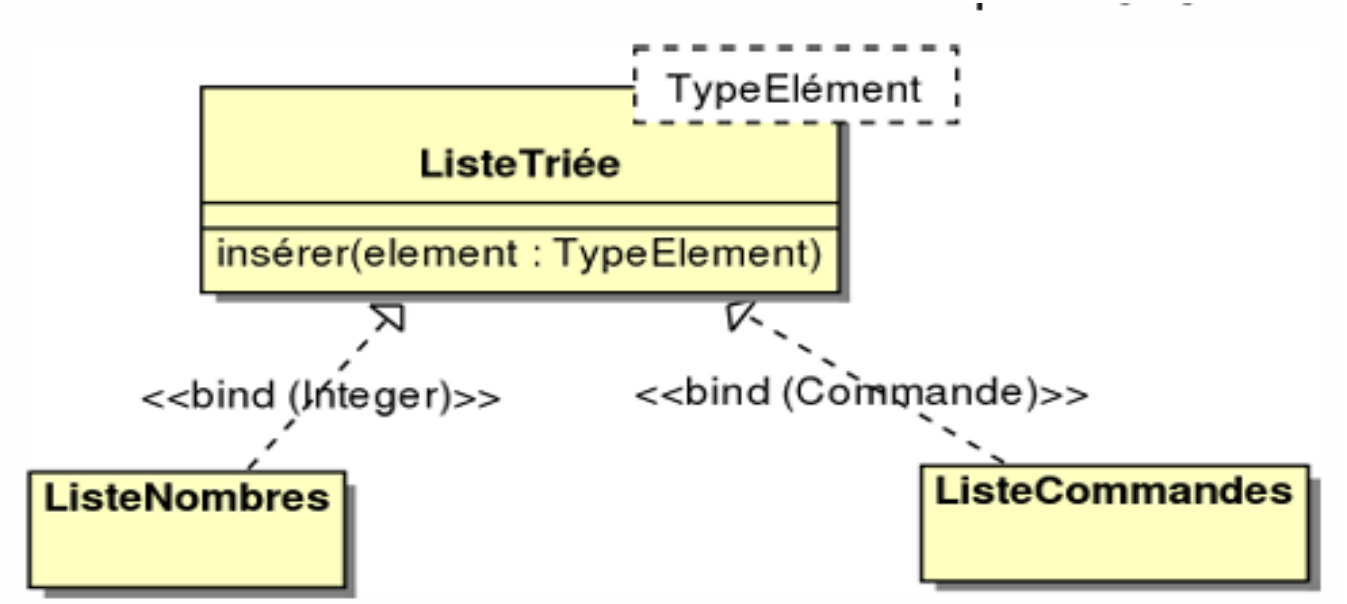


Diagramme de classes

- **Construction d’un diagramme de classe**

1. Identifier les classes du domaine étudié, souvent basées sur des concepts et des substantifs.
2. Définir les associations entre les classes, généralement exprimées par des verbes liant plusieurs classes.
3. Identifier les attributs des classes, souvent des substantifs représentant des détails plus fins que les classes. Les adjectifs et valeurs correspondent souvent à des attributs.
4. Organiser et simplifier le modèle en utilisant l'héritage.
5. Tester les relations entre les classes.
6. Répéter et affiner le modèle.