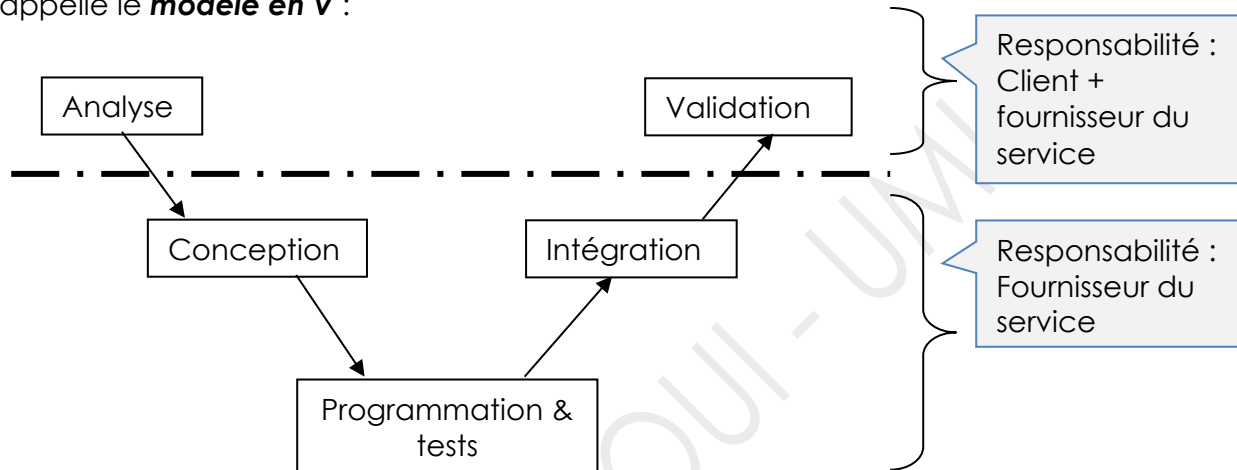


- **Validation** : Vérifier si le produit est conforme aux besoins exprimés dans le cahier des charges
- **Documentation & formation** : Rédiger les documents nécessaires à l'utilisation du logiciel, et former les prochains utilisateurs sur ce dernier si c'est nécessaire
- **Livraison** : Installation du livrable chez le client

La phase de réalisation est l'étape concrète de création du logiciel.

Le cycle de vie de ce dernier est souvent représenté dans cette phase par ce qu'on appelle le **modèle en V** :



B. Pourquoi modéliser

La conception, ou modélisation, consiste à schématiser, ou créer une représentation virtuelle, des besoins exprimés dans le cahier des charges.

Cela permet de :

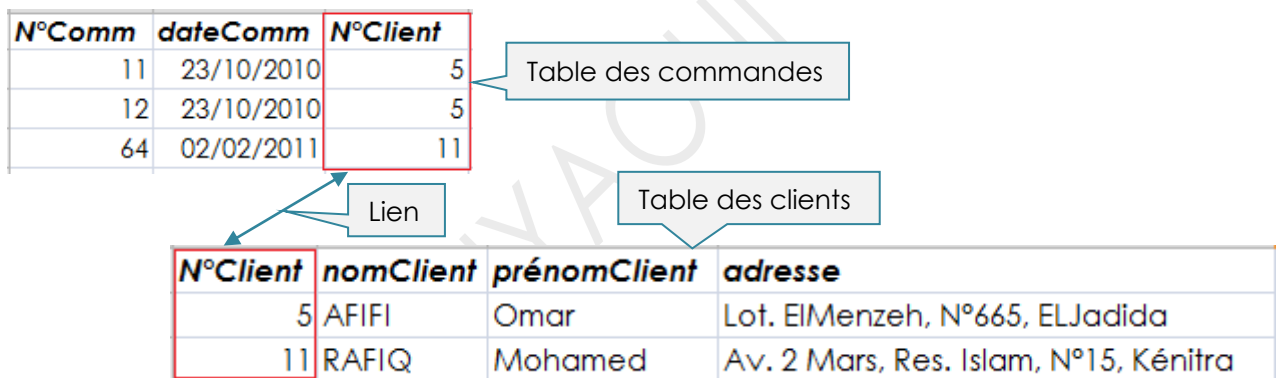
- Minimiser au maximum les erreurs lors de la phase de programmation, car durant cette phase, les erreurs sont beaucoup plus coûteuses en temps, donc en argent.
- Assurer la cohérence et éviter la redondance des données
- Assurer la conformité du logiciel aux exigences du client

Exemple : Gestion de commandes

N°Comm	dateComm	N°Client	nomClient	prénomClient	adresse
11	23/10/2010	5	AFIFI	Omar	Lot. ElMenzeh, N°665, ELJadida
12	23/10/2010	5	AFIFI	Omar	Lot. ElMenzeh, N°665, ELJadida
64	02/02/2011	11	RAFIQ	Mohamed	Av. 2 Mars, Res. Islam, N°15, Kénitra

Pour **2** commandes, les informations du client **N°5** sont saisies **2** fois.

Il serait plus logique d'avoir deux tableaux avec un lien entre les deux :



Les questions qui se posent donc sont :

- Quel champ doit être placé dans quelle table ?
- Quel est le champ qui jouera le rôle du lien entre les tables ?

Pour répondre à ce genre de questions, il est nécessaire de recourir à une solution d'analyse et de conception qui soit **méthodique et standardisée**. Les plus utilisées sont **MERISE** et **UML**.

C. Introduction à UML

1. Définition

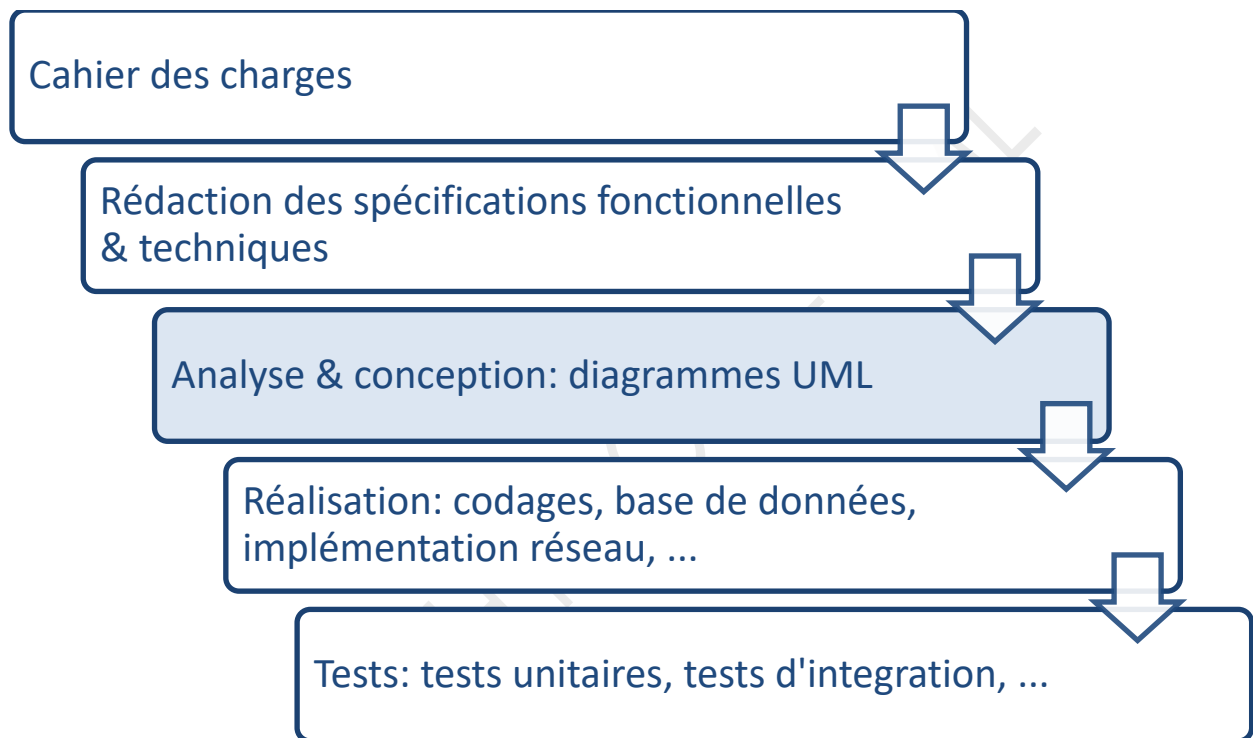
UML (*Unified Modeling Language*) est un langage, ou notation, de modélisation graphique à base de diagrammes, utilisé dans la conception & développement orientés objet.

UML est un langage standardisé par l'*Object Management Group*

(<http://www.uml.org>)

La dernière version diffusée par l'OMG est UML 2.4.1 depuis août 2011.

2. Quand utiliser UML



3. Aperçus sur les diagrammes d'UML

Les diagrammes UML se répartissent en **2** grands groupes :

- Diagrammes **structurels ou statiques**
- Diagrammes **comportementaux**

Diagrammes structurels ou statiques

- Diagramme de classes (Class diagram)
- Diagramme d'objets (Object diagram)
- Diagramme de composants (Component diagram)
- Diagramme de déploiement (Deployment diagram)
- Diagramme des paquetages (Package diagram)
- Diagramme de structure composite (Composite structure diagram)
- Diagramme de profils (Profil Diagram)

Diagrammes comportementaux

- Diagramme des cas d'utilisation (use-cases ou Use Case Diagram)
- Diagramme d'états-transitions (State Machine Diagram)
- Diagramme d'activité (Activity Diagram)
- Diagramme de séquence (Sequence Diagram)
- Diagramme de communication (Communication Diagram)
- Diagramme global d'interaction (Interaction Overview Diagram)
- Diagramme de temps (Timing Diagram)

(Les 4 derniers sont souvent appelés *Diagrammes d'interaction ou dynamiques*)

Grâce aux outils de modélisation UML, il est également possible de générer automatiquement des parties de code (*par exemple en langage Java*) ou des supports de documentation à partir des modèles réalisés.

II. Diagrammes d'analyse

A. Modélisation des besoins : Diagramme des cas d'utilisation

1. Définition

Définition : Besoins fonctionnels / techniques

Tirés du cahier des charges, les **besoins** ou **exigences** sont l'expression documentée de ce qu'une application informatique **devrait faire** ou **doit être**

Deux types d'exigences :

→ **Exigences fonctionnelles**

Décrivent le système sur le plan **fonctionnel**, c.à.d. ce que le système doit faire.

Elles reflètent les besoins exprimés par le client dans le cahier de charges, et elles ne sont pas négociables.

En général, ce genre d'exigences répond à la question : « Pourquoi l'utilisateur aurait-il besoin de l'application ? »

→ **Exigences non fonctionnelles (ou techniques)**

Décrivent le système sur le plan **technique**, c.à.d. la manière dont il exécute les fonctionnalités que l'utilisateur lui demande

Elles reflètent généralement les contraintes techniques imposées par l'architecture logicielle du futur produit : le choix de la plateforme et du langage de programmation, le type de système d'exploitation visé (*Unix, Windows, Android, ...*), le déploiement sur le réseau (*machine locale, intranet, internet, ...*), etc.

Ce genre d'exigences peut être exprimé par le client ou par le développeur

Exemple : Besoins fonctionnels / techniques

Etude de cas : Sites internet pour réseaux sociaux

- Pouvoir ouvrir plusieurs conversations instantanées à la fois : figure parmi les services les plus élémentaires et indispensables d'une application de réseau social, ce service doit donc être considéré comme une **exigence fonctionnelle**

- Pouvoir ouvrir plus de 50 conversations instantanées à la fois : ceci peut être réalisable ou non, selon l'architecture technique de l'application (plateforme, environnement de développement, langage de programmation, ...), ce service doit donc être considéré comme une **exigence non fonctionnelle**

2. Analyse des besoins

Exemple : Besoins fonctionnels / techniques

Etude de cas : Guichet automatique bancaire (GAB)

A partir du texte du cahier de charges de l'application du GAB, on peut déduire les exigences suivantes :

- **Exigences fonctionnelles**

- L'utilisateur (client de la banque) doit pouvoir consulter le solde de son compte
- L'utilisateur doit pouvoir extraire un mini-relevé de son compte
- L'utilisateur doit pouvoir retirer de l'argent de son compte
- ...

- **Exigences non fonctionnelles**

- L'opération d'extraction d'un mini-relevé de compte bancaire ne doit pas durer plus de 3sec
- L'impression du reçu d'envoi de demande de chéquier ne doit pas durer plus de 5sec
- ...

Etude de cas : Gestion commerciale

[.....

Pour passer une commande le client s'adresse au réceptionniste, celui-ci passe la commande (commande normale ou express) si le client existe déjà, sinon il doit d'abord le créer.

Le comptable ou le directeur des ventes peuvent enregistrer la facture de chaque commande dont le paiement est dûment réglé.

Le livreur doit enregistrer chaque livraison qu'il effectue.

Pour les commandes dont la liste de produits n'existe pas en quantité suffisante, le directeur des ventes peut aussi gérer le stock pour éviter toute rupture de produits.

La gestion du stock contient les fonctionnalités suivantes :

Gérer un produit, et enregistrer une demande d'alimentation du stock

La gestion d'un produit contient les fonctionnalités suivantes :

Ajouter, supprimer et mettre à jour un produit

Chaque utilisateur du système, doit d'abord s'authentifier avant de pouvoir accéder à n'importe quelle fonctionnalité.

.....]

Travail à faire :

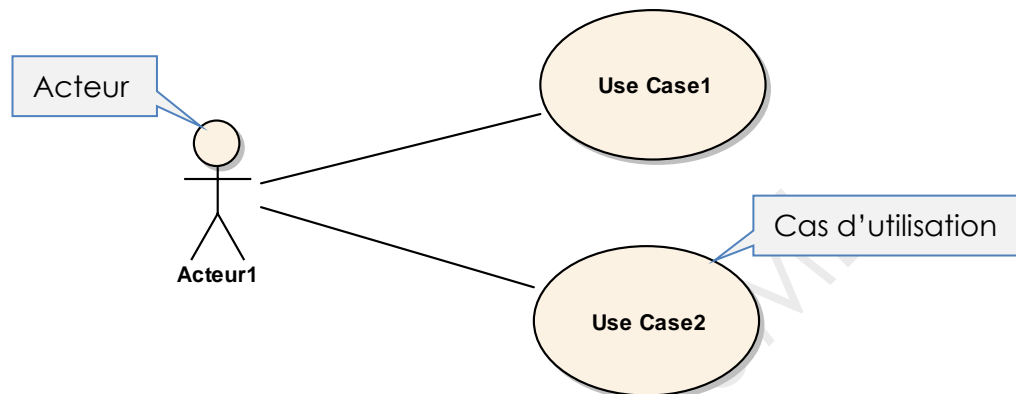
- **Rédiger la liste des exigences fonctionnelles**

3. Modélisation des besoins : Diagramme des Cas d'utilisation

Définition : Diagramme des cas d'utilisation

Le diagramme des cas d'utilisation – *use case diagram* – est considéré comme le point d'entrée de la phase d'analyse, il permet de modéliser toutes les fonctionnalités que doit fournir le système.

Le formalise graphique du DCU est principalement basé sur les **acteurs** et les **cas d'utilisation**

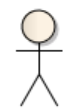


Chaque exigence ou besoin **fonctionnel** identifié dans la phase de rédaction des exigences, est représenté par un use-case.

4. Réaliser le UCD

→ Les acteurs

Définition : Les acteurs



Les **acteurs** sont soit (le plus souvent) des utilisateurs humains du système, soit d'autres systèmes informatiques ou hardwares qui vont communiquer avec le système.

Exemples :

Gestionnaire de stock, chargé de clientèle, administrateur, ...

Exemple

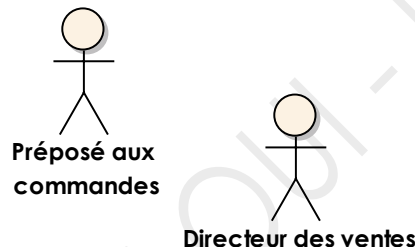
Fragment d'un cahier de charges : Gestion commerciale

[...

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

Par contre, un simple préposé aux commandes n'a pas le droit d'accéder à la gestion du stock

...]



→ Les relations entre acteurs

La seule relation possible entre deux acteurs est le lien d'héritage (ou généralisation) :

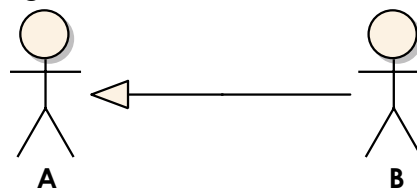
Définition : L'héritage entre acteurs

On dit qu'un acteur **A** est une **généralisation** d'un acteur **B**, on dit aussi : l'acteur **B** est un cas particulier de l'acteur **A**, quand :

Tous les cas d'utilisation appartenant à l'acteur **A**, appartiennent aussi à **B**

On dit alors que **B hérite de A**

Notation graphique de l'héritage :



Exemple

Etude de cas : Gestion commerciale

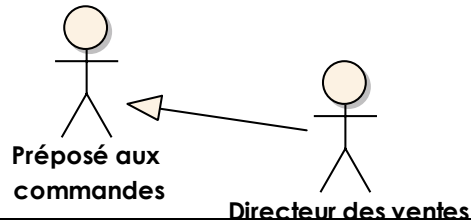
[...

Le préposé aux commandes peut passer une commande, et suivre une commande. Le directeur des ventes **est lui aussi** un préposé aux commandes, mais avec un **pouvoir supplémentaire** : en plus de pouvoir passer et suivre une commande, il peut gérer le stock.

Par contre, un simple préposé aux commandes n'a pas le droit d'accéder à la gestion du stock

...]

Acteurs :



→ Les cas d'utilisation

Définition : Les cas d'utilisation

Chaque **cas d'utilisation** représente **une exigence fonctionnelle**.

Exemples :

Retirer argent, Passer une commande, Supprimer un produit, ...

Exemple**Cas d'utilisation : Retirer argent****Etude de cas : GAB**

Retirer argent

Un cas d'utilisation doit **obligatoirement** avoir les éléments suivants :

- Un nom

Le nom d'un cas d'utilisation commence généralement par un verbe à l'infinitif

**IMPORTANT**

- Deux ou plusieurs cas d'utilisation ne peuvent avoir le même nom.
- Un cas d'utilisation ne peut avoir plusieurs noms.

- Un ou plusieurs acteurs déclencheurs

Un use-case doit avoir au moins un seul acteur déclencheur.

Définition : Acteur déclencheurUn acteur déclencheur d'un use-case **A** est un acteur qui utilise le système pour accéder à la fonctionnalité représentée par le cas **A**.**Exemple****Etude de cas : Gestion de centre de formation**

[...

Le directeur pédagogique peut ajouter ou supprimer des modules tout au long de l'année scolaire

...]

On peut donc dire que l'acteur **directeur pédagogique** est l'acteur déclencheur des cas d'utilisation **ajouter un module** et **supprimer un module****REMARQUES**

- Un acteur peut posséder plusieurs cas d'utilisation.
- Un use-case peut avoir plusieurs acteurs déclencheurs.

- Des scénarios

Un cas d'utilisation doit obligatoirement avoir un scénario principal (**unique**), et peut avoir un ou plusieurs scénarios secondaires.

Définition : Scénario / Scénario principal / Scénario secondaire

- **Scénario** : décrit (sous forme de texte) les échanges d'évènements entre l'acteur et le système.
- **Scénario principal** : représente le déroulement de la réalisation avec succès du use-case. Il doit se terminer par la fin attendue de la part de l'acteur déclencheur.
- **Scénario d'erreur (ou d'exception)** : permet de préciser la conduite à tenir par le système en cas d'anomalies ou d'erreurs.
- **Scénario alternatif** : scénario sans erreur mais qui décrit une fin alternative à celle du scénario principal.

Exemple

Etude de cas : GAB

Cas d'utilisation : « Retirer argent »

Scénario principal : Argent retiré

Scénario d'erreur N°1 : Carte invalide,

fin du scénario : carte éjectée

Scénario d'erreur N°2 : Mot de passe invalide, fin du scénario : carte aspirée

Scénario d'erreur N°3 : Caisse épuisée, fin du scénario : carte éjectée

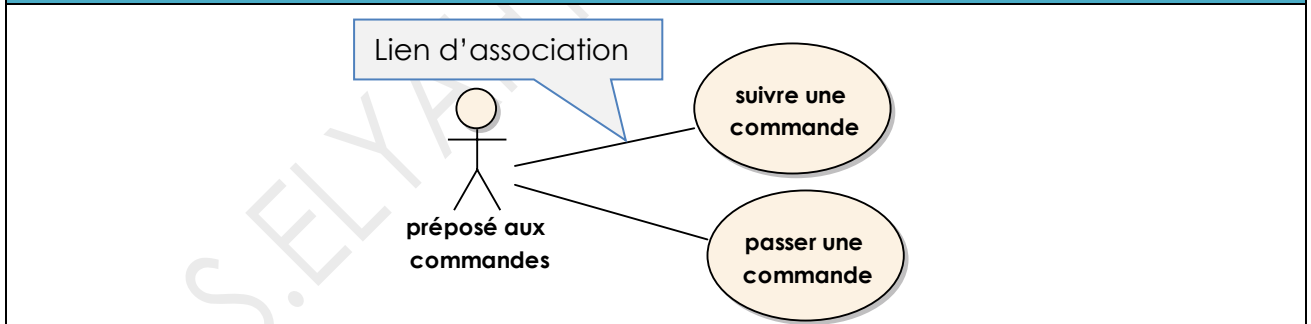
Scénario d'erreur N°4 : Solde insuffisant, fin du scénario : carte éjectée

→ La relation d'association entre acteur et cas d'utilisation

Chaque acteur **doit être lié** par un lien d'association avec tous les use-cases qu'il déclenche, ce lien est représenté par un trait **continu**

Exemple

Etude de cas : Gestion commerciale



Etude de cas : Guichet automatique bancaire

[.....

Le porteur de carte (soit client de la banque soit client d'une autre banque) peut retirer de l'argent.

Si le porteur est un client de la banque, il peut aussi consulter son compte, ou retirer un mini-relevé bancaire.

Il peut aussi (client de la banque) modifier ses paramètres de sécurité
.....]

Travail à faire :

- Identifier les acteurs et leurs cas d'utilisation associés
- Réaliser le diagramme des cas d'utilisation

→ **Les relations entre cas d'utilisation**

UML définit **3** principaux types de relations entre cas d'utilisation :

- L'héritage

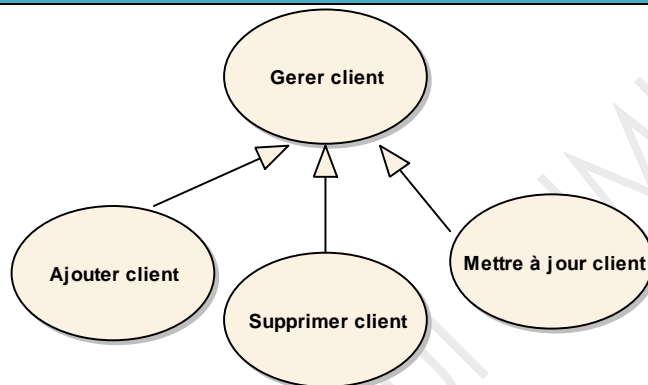
Définition

On dit qu'un use-case **B** hérite d'un autre use-case **A**, quand **A** représente une généralisation de **B**

Graphiquement, l'héritage entre cas d'utilisation est noté de la même manière que celui des acteurs

Exemple

Etude de cas : Gestion commerciale

**IMPORTANT**

- L'association d'héritage entre use-cases est utilisée principalement dans un souci d'organisation graphique du UCD
- En général, un use-case générique (dans l'exemple : « *Gérer client* ») n'a pas d'existence propre à lui, d'ailleurs il ne peut pas avoir de scénarios, il ne se réalise qu'à travers l'un de ses use-case dérivés

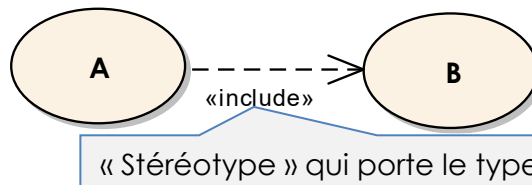
- L'inclusion

Définition : « include »

On dit qu'un use-case **B** est inclus d'un autre use-case **A**, quand le déroulement de **B** représente **toujours** une partie du déroulement de **A**

Autrement dit, si **B** est inclus dans **A**, alors l'acteur **ne peut réaliser le use-case A, que si B est terminé correctement**.

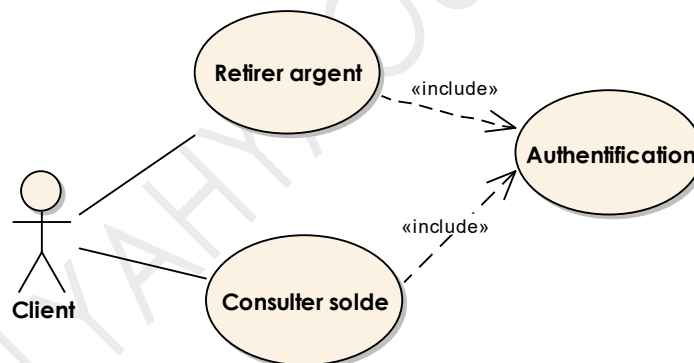
Graphiquement, la relation d'inclusion est notée par une flèche discontinue allant de **A** à **B**, accompagnée du stéréotype « **include** »

**Exemple****Etude de cas : GAB**

[...

Pour pouvoir retirer de l'argent ou simplement consulter son solde, un client doit **toujours** s'authentifier en insérant sa carte guichet, le système analyse la carte et demande le mot de passe ...

...]

**REMARQUES**

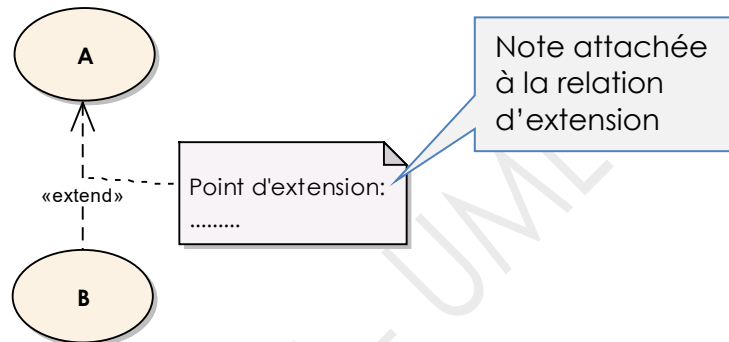
- Si un use-case **B** est inclus dans un autre **A**, alors en général **B** n'est pas à proprement dit **un vrai cas d'utilisation**, car pour l'utilisateur du logiciel le cas d'utilisation **B** ne représente pas une fonctionnalité du système, donc le fait d'associer l'acteur au use-case **B** serait **une erreur**.
- Une autre erreur courante est d'utiliser la relation « *include* » pour faire « le découpage » d'un cas d'utilisation en plusieurs « sous cas d'utilisation » qui s'enchaînent en fonction de certains critères.

- L'extension

Définition : « extends »

On dit qu'un use-case **B** étend un autre use-case **A**, quand **B** peut être appelé au cours de l'exécution de **A** si une condition est vérifiée (**donc pas toujours**). Cette condition est appelée **point d'extension**.

Graphiquement, la relation d'inclusion est notée par une flèche discontinue allant de **B** à **A**, accompagnée du stéréotype « **extends** », cette flèche **doit obligatoirement** être commentée par une note indiquant le point d'extension



i REMARQUE

On peut attacher une note à n'importe quel élément graphique dans un diagramme UML

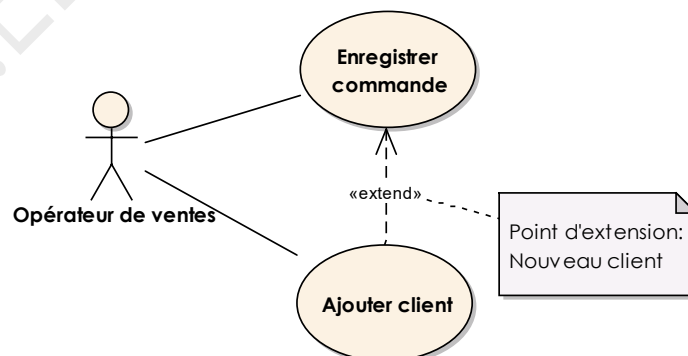
Exemple 1**Etude de cas : Gestion commerciale**

[...

L'opérateur de ventes peut enregistrer les commandes des clients, et chaque commande doit appartenir à un client parmi la liste des clients enregistrés dans le système.

S'il s'agit d'un nouveau client, l'opérateur doit pouvoir le créer avant de lui enregistrer sa commande, ...

...]

**Exemple 2****Etude de cas : Gestion électronique des actes de ventes immobilières**

[...

Le notaire doit pouvoir disposer d'une nouvelle fenêtre pour la création de chaque contrat.

Chaque contrat contient une liste de vendeurs et une liste d'acquéreurs, le notaire doit donc pouvoir choisir un client parmi la liste des clients (si le client existe déjà), et l'ajouter

soit dans la liste des vendeurs, soit dans celle des acquéreurs.

S'il s'agit d'un nouveau client, il doit d'abord le créer.

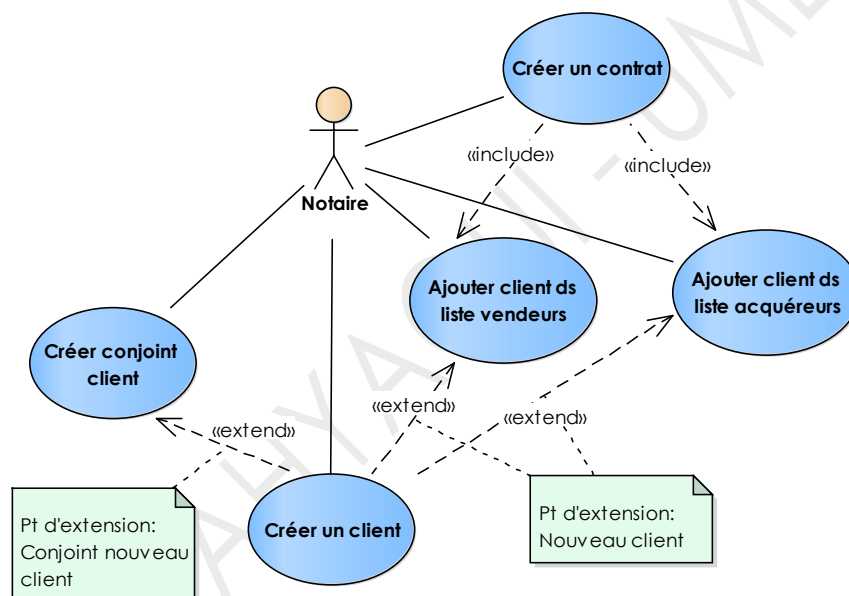
Lors de la création d'un client, (si le client est marié) le notaire doit pouvoir spécifier les informations relatives au conjoint du client. Si le conjoint du client n'existe pas, il doit d'abord le créer.

...]

Exigences fonctionnelles :

- Créer un contrat
- Ajouter un client dans liste des vendeurs
- Ajouter un client dans liste des acquéreurs
- Créer un client
- Créer le conjoint d'un client

Diagramme des cas d'utilisation :



Etudes de cas :

Guichet automatique bancaire

[.....

N'importe quel porteur de carte bancaire peut retirer de l'argent, ou (s'il le souhaite) retirer de l'argent avec un reçu.

Si le porteur est un client de la banque, il doit pouvoir accéder aussi aux services suivants (en plus du retrait d'argent) :

Déposer de l'argent, retirer un mini-relevé bancaire de son compte, consulter son solde, payer la facture d'abonnement du téléphone fixe et modifier ses paramètres de sécurité.

Un porteur de carte ne peut accéder à aucun service du GAB sans être d'abord authentifié.

.....]

Travail à faire :

- **Identifier les acteurs et leurs cas d'utilisation associés**
- **Réaliser le diagramme des cas d'utilisation**

Gestion commerciale

[.....

Pour passer une commande le client s'adresse au réceptionniste, celui-ci passe la commande (commande normale ou express) si le client existe déjà, sinon il doit d'abord le créer.

Le comptable et le directeur des ventes peuvent enregistrer la facture de chaque commande dont le paiement est dûment réglé.

Le livreur doit enregistrer chaque livraison qu'il effectue.

Pour les commandes dont la liste de produits n'existe pas en quantité suffisante, le directeur des ventes peut aussi gérer le stock pour éviter toute rupture de produits.

La gestion du stock contient les fonctionnalités suivantes :

Gérer un produit, et enregistrer une demande d'alimentation du stock

La gestion d'un produit contient les fonctionnalités suivantes :

Ajouter, supprimer et mettre à jour un produit

Chaque utilisateur du système, doit d'abord s'authentifier avant de pouvoir accéder à n'importe quelle fonctionnalité.

.....]

Travail à faire :

- **Identifier les acteurs et leurs cas d'utilisation associés**
- **Réaliser le diagramme des cas d'utilisation**

Gestion de réservations d'hôtel

[.....

La direction d'un hôtel souhaite créer un système informatique pour la gestion des chambres et des réservations.

Ce système sera composé de deux parties, un site web pour les réservations des clients par internet, et une application Windows pour les employés de l'hôtel.

Un visiteur normal du site peut uniquement afficher la liste des chambres.

S'il s'agit d'un client, il peut modifier ses informations (login, password, nom, numéro de téléphone, ...), afficher la liste des chambres, passer une réservation ou bien annuler sa réservation.

Au cas où c'est le réceptionniste qui enregistre la réservation pour le client, il (réceptionniste) doit d'abord ajouter le client si ce dernier n'est pas encore enregistré dans le système.

De même, si c'est un nouveau client qui passe la réservation, il doit d'abord créer un compte.

Le réceptionniste peut également afficher la liste des chambres disponibles, la liste des réservations, et la liste des clients.

Le gérant quant à lui, peut en plus de cela, gérer la liste des clients.

La gestion de la liste des clients comprend l'ajout, la mise à jour, et la suppression de clients.

Chaque utilisateur du système doit s'authentifier, sauf les visiteurs normaux, pour l'affichage de la liste des chambres.

.....]

Travail à faire :

- Identifier les acteurs et leurs cas d'utilisation associés
- Réaliser le diagramme des cas d'utilisation

Gestion de médiathèque

[.....

Le système d'informations d'une médiathèque de livres informatiques comporte deux interfaces graphiques : un site internet pour les clients, et une application Windows pour les bibliothécaires et l'administrateur.

Le site internet est ouvert à tout internaute pour parcourir la liste des livres, afficher le sommaire d'un livre ou encore, créer un compte d'adhérent.

Si l'internaute est un adhérent, il peut aussi (après s'être authentifié) créer un panier, enregistrer un panier pour une modification ultérieure, ouvrir un panier déjà créé, valider l'achat d'un panier (un panier contient au moins un livre).

Il peut aussi (l'adhérent) modifier les informations de son compte (nom, prénom, ...) ou signaler au système qu'il a oublié son mot de passe.

Le bibliothécaire (utilisateur de l'interface Windows) doit pouvoir ajouter ou supprimer un livre, ajouter ou supprimer une catégorie de livres (développement, base de données, systèmes & réseaux, ...), ou encore afficher la liste des paniers validés, et la liste des paniers livrés.

Un livre ne doit pas exister dans la médiathèque, sans appartenir à une catégorie.

Donc lors de l'ajout d'un livre, sa catégorie doit être spécifiée. S'il s'agit d'une nouvelle catégorie, elle doit être d'abord créée.

L'administrateur (lui aussi utilisateur de l'interface Windows) doit pouvoir afficher tous les comptes utilisateurs (qu'ils soient de type adhérent ou bibliothécaire), supprimer ou créer un compte bibliothécaire, ou modifier les informations de son propre compte à lui (nom, prénom, mot de passe, ...).

Il doit aussi (l'administrateur) pouvoir afficher la liste des paniers validés, et la liste des paniers livrés, et afficher la recette du jour et la recette du mois.

*Bibliothécaires et administrateur doivent aussi s'authentifier.
.....]*

Travail à faire :

- **Identifier les acteurs et leurs cas d'utilisation associés**
- **Réaliser le diagramme des cas d'utilisation**

B. Diagramme de séquence

1. Introduction

D'après les thèmes exposés précédemment, le use-case diagram ne fournit qu'une description « quantitative » des cas d'utilisation, sans se préoccuper de la chronologie et de la manière de déroulement de chaque use-case. C'est plutôt le « diagramme de séquence » qui a été prévu à cet effet.

2. Scénarios d'un cas d'utilisation

La rédaction des scénarios d'un use-case doit se faire de manière textuelle brève, claire, et organisée de manière chronologique.

Exemple

Etude de cas : Gestion commerciale

Scénarios du use-case : Authentification

Scénario principal :

1. L'acteur tape son login et son mot de passe
2. Le système analyse le login et le mot de passe
3. Le système affiche un message d'accueil

Scénario d'erreur N°1 : Login ou mot de passe invalide

1. L'acteur tape son login et son mot de passe
2. Le système analyse le login et le mot de passe
3. Le système affiche un message d'erreur

3. Diagramme de séquence d'analyse

Définition

Le **diagramme de séquences d'analyse**, appelé aussi **diagramme de séquences système**, décrit de manière graphique **tous les scénarios** du cas d'utilisation, en modélisant les échanges entre l'acteur et le système, et en tenant compte du facteur temporel.



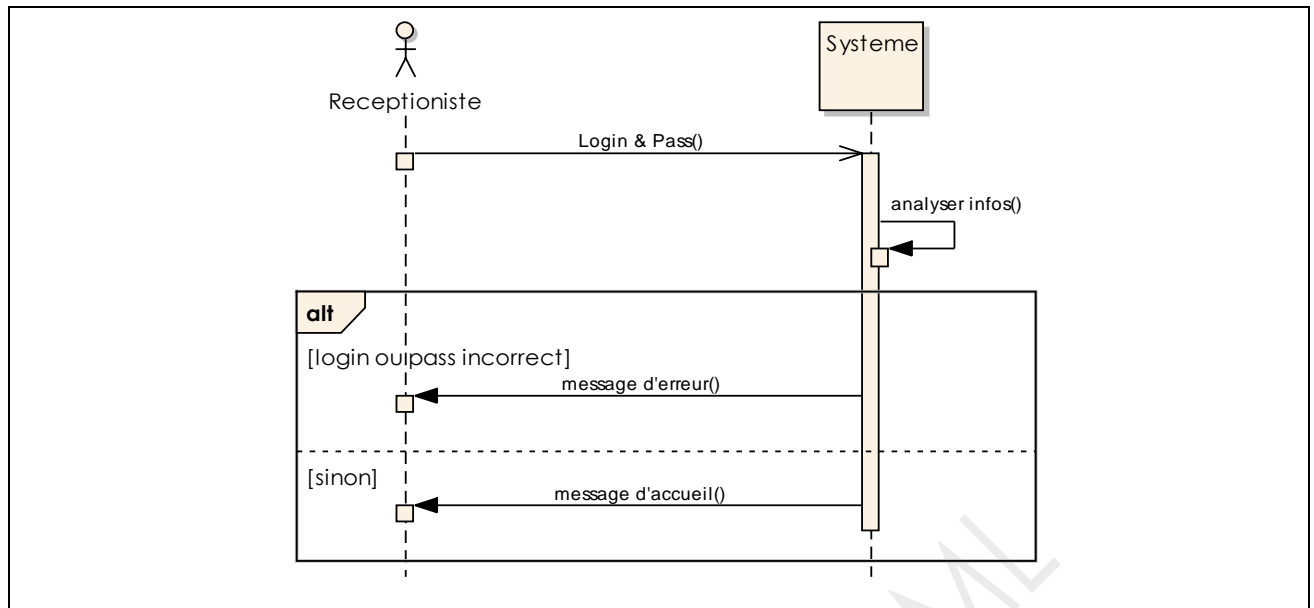
IMPORTANT

- Quel que soit le nombre de ses scénarios, un cas d'utilisation ne peut avoir qu'un seul diagramme de séquence
- Dans un diagramme de séquence système, les seuls intervenants sont l'acteur et le système, représenté comme une boîte noire, sans détailler ce qui se passe à l'intérieur du système

Exemple

Etude de cas : Gestion commerciale

Diagramme de séquence système : Authentification (Acteur : Réceptionniste)



4. Réaliser le diagramme de séquence d'analyse

Le formalisme graphique du diagramme de séquence est principalement basé sur les éléments suivants : **Les lignes de vie**, **les messages** et **les fragments**.

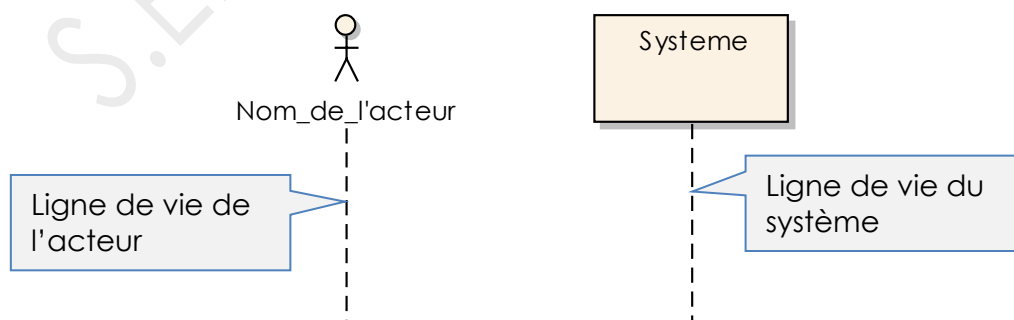
→ Diagramme de séquence : Les lignes de vie (Lifeline)

- Les lignes de vie

Définition

Une ligne de vie représente la ligne chronologique sur-laquelle se déroule l'ensemble des actions effectuées par l'acteur ou le système

Les lignes de vie sont représentées graphiquement par une ligne verticale discontinue, comme ceci :



- Les lignes de vie : Période d'activité

Définition : Période d'activité

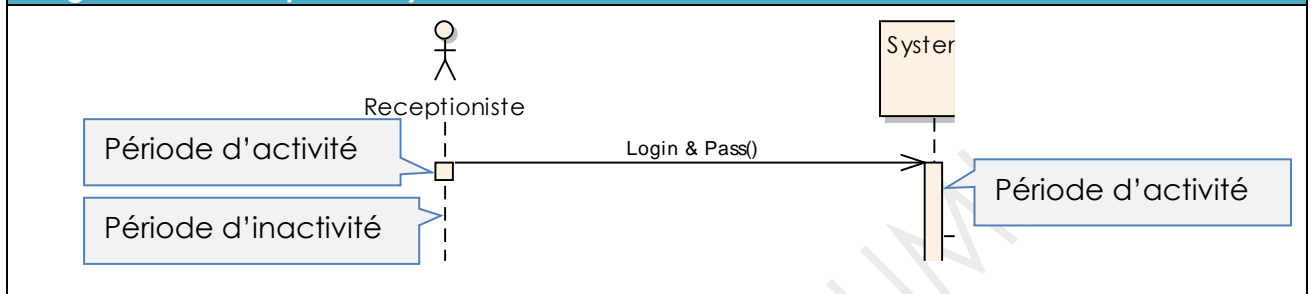
Une ligne de vie n'est pas toujours active. On doit donc représenter de manière explicite ses périodes d'activité.

Ces périodes sont représentées au moyen d'une bande rectangulaire superposée à la ligne de vie.

Exemple

Etude de cas : Gestion commerciale

Diagramme de séquence système : Authentification



→ Diagramme de séquence : Les messages

Définition

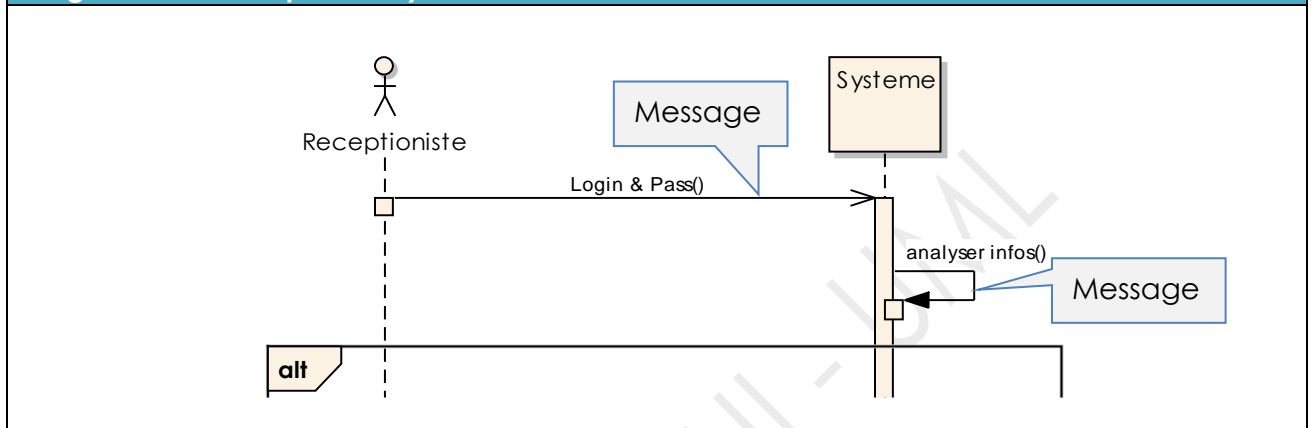
Dans un diagramme de séquence, chaque interaction entre l'acteur et le système est un message.

Un message est représenté par une flèche dont le départ est la ligne de vie de l'objet émetteur, et la terminaison est la ligne de vie de l'objet récepteur

Exemple

Etude de cas : Gestion commerciale

Diagramme de séquence système : Authentification



▪ Les Types de messages

- Message synchrone / asynchrone

Définition : Message synchrone

L'expéditeur reste inactif (sans pouvoir envoyer d'autres messages) jusqu'à ce que le destinataire réponde.

Ce genre de message – parfois appelé aussi **call**, ou **appel** – correspond à une demande de la part est souvent associé à l'appel de méthodes entre des objets de classes différentes

(La notion d'appel de méthode sera détaillée dans le chapitre Conception)

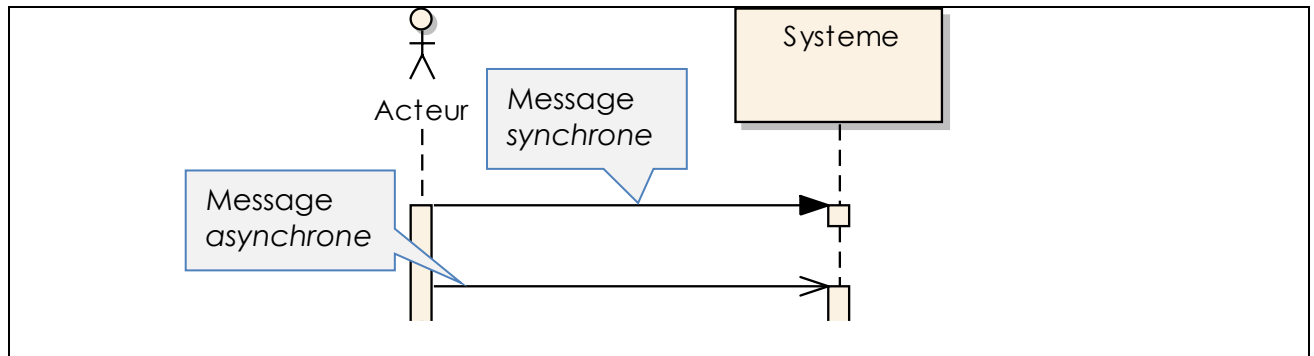
Le message synchrone est modélisé par **une flèche à tête pleine**, allant de la ligne de vie de l'expéditeur vers celle du destinataire

**Définition : Message asynchrone**

L'expéditeur n'attend pas la réponse du destinataire pour pouvoir envoyer d'autres messages. Ce genre de message est parfois appelé aussi **signal**

Le message asynchrone est modélisé par **une flèche simple**, allant de la ligne de vie de l'expéditeur vers celle du destinataire





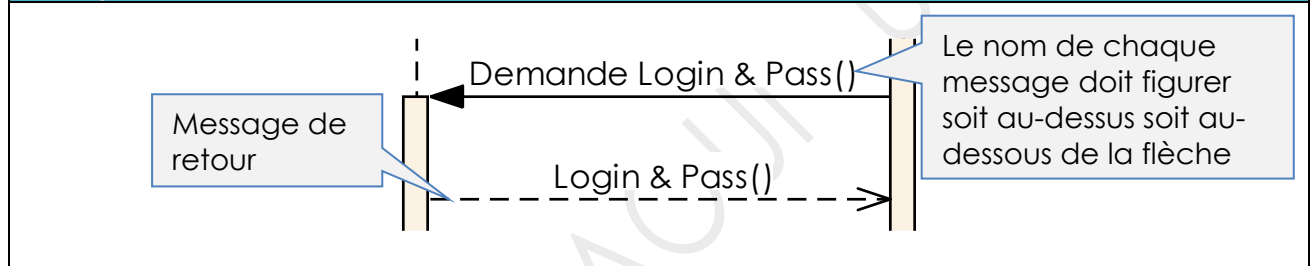
○ **Message de retour**

Définition

C'est un message qui est immédiatement envoyé en réponse au message précédent. Ce type de message est modélisé par **une flèche discontinue**



Exemple



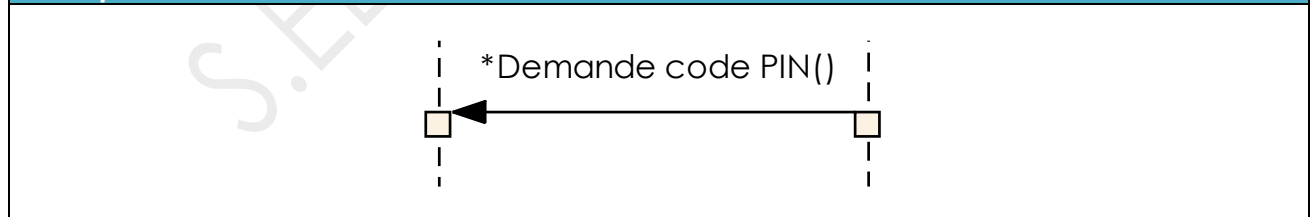
○ **Message itératif**

Définition

C'est un message qui se répète (*genre de messages utilisé généralement pour modéliser les boucles*).

Le nom du message **doit** être précédé par une étoile « * ».

Exemple



o Message réflexif

Définition

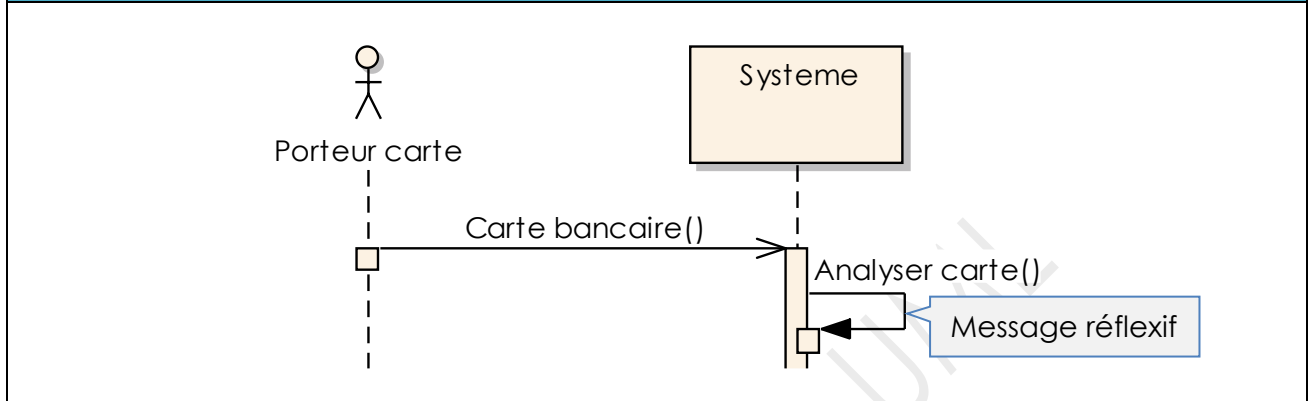
Message dont l'expéditeur est lui-même récepteur.

Exemple

Etude de cas : G.A.B

Diagramme de séquence système : Authentification

Message réflexif



o Message conditionné

Définition

Message qui n'est envoyé que si une condition est réalisée. Cette condition est appelée **condition de garde**.

La condition de garde est toujours exprimée entre deux crochets qui doivent précéder le nom du message.



Exercice :

Guichet automatique bancaire (GAB)

Travail à faire :

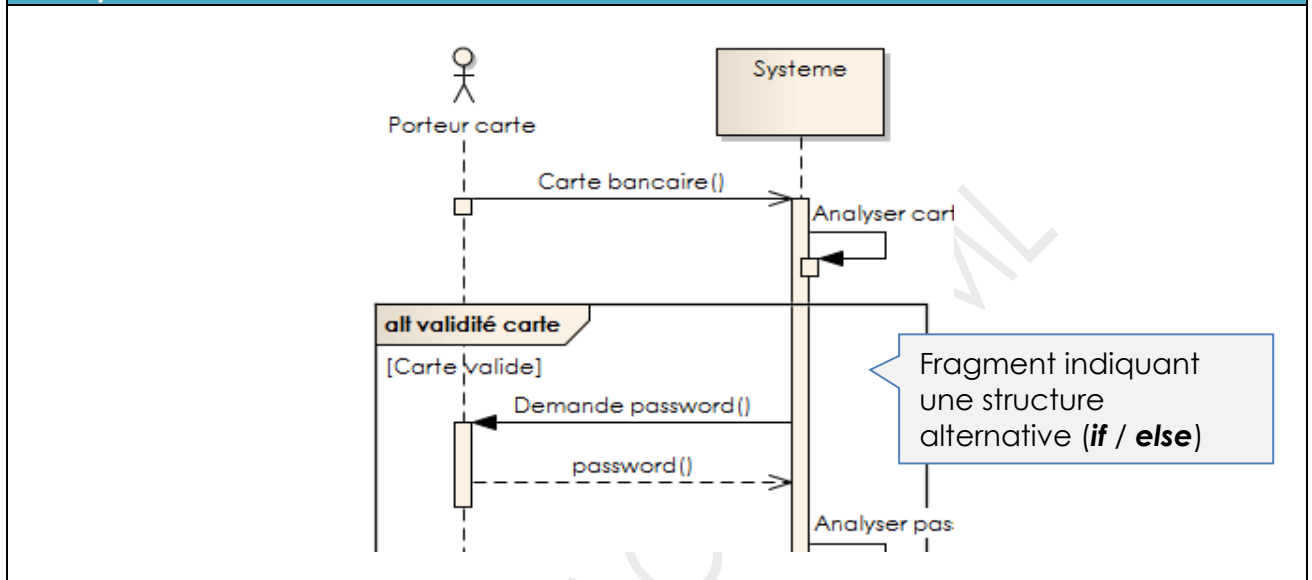
- Réaliser les scénarios du use-case *Authentification*
- Réaliser le diagramme de séquences de ce cas d'utilisation

→ Diagramme de séquence : Les fragments

Définition : Fragments

Un fragment est un cadre rectangulaire inséré dans le déroulement d'un diagramme de séquence. Les fragments servent principalement à exprimer :

- une relation d'inclusion/extension entre deux use-cases
- une structure alternative (**if**, **if/else**, **switch**, ...)
- une structure itérative (**for**, **while**, **loop**, ...)

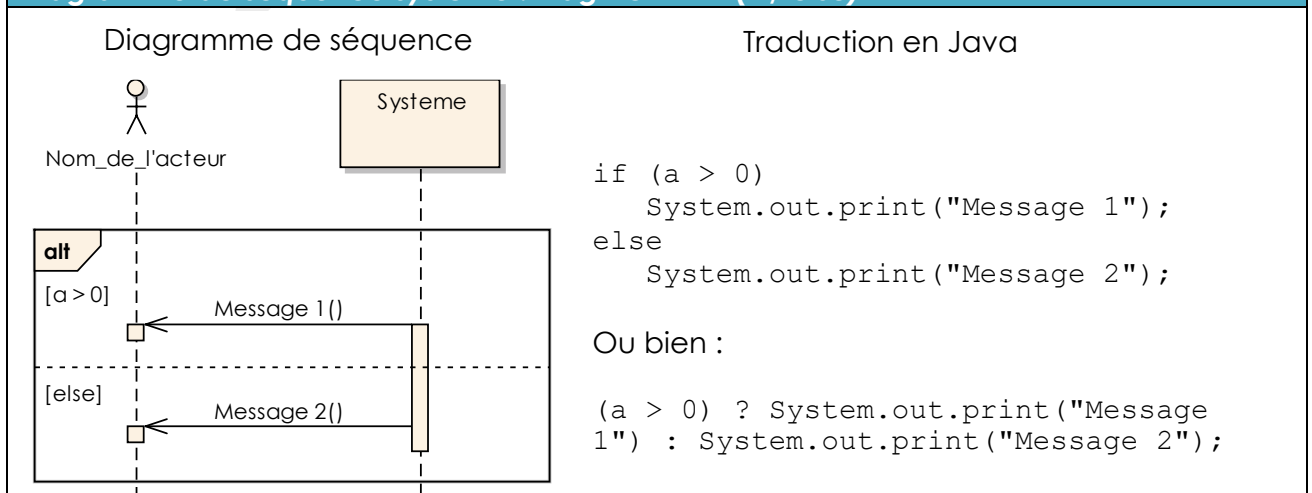
Exemple

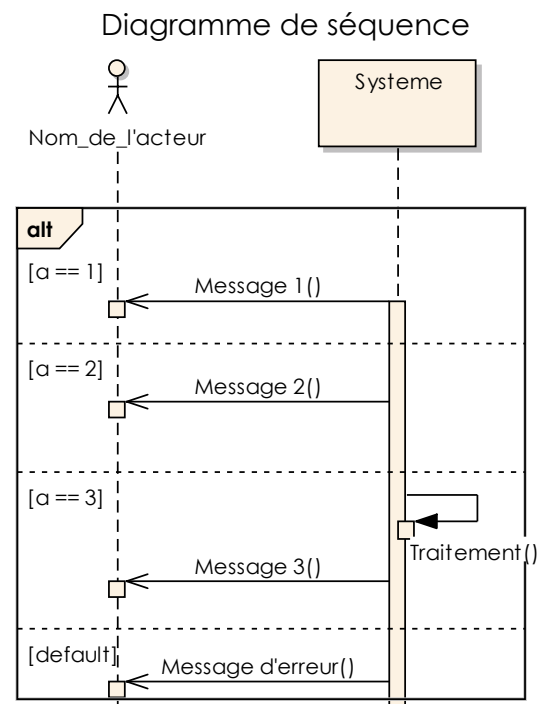
▪ Fragments : Les structures alternatives

Définition : Structures alternatives

Les structures alternatives sont représentées – selon leur nombre d'alternatives – en UML par deux types de fragments :

- structures ayant une seule alternative (équivalent en Java : **if**), représentées par le fragment **OPT**
- structures ayant 2 ou plusieurs alternatives (équivalent en Java : **?:**, **if/else**, **switch**), représentées par le fragment **ALT**

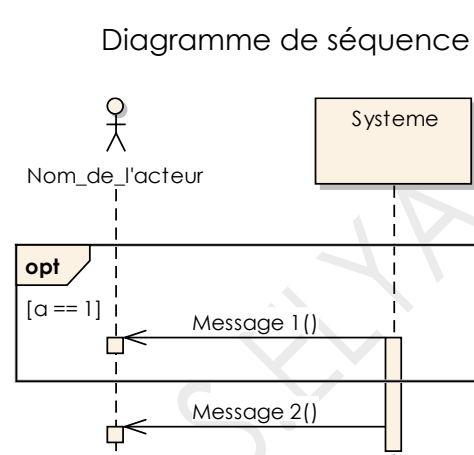
Exemple 1**Diagramme de séquence système : Fragment ALT (if / else)**

Exemple 2**Diagramme de séquence système : Fragment ALT (switch)**

Traduction en Java

```

switch(a) {
    case 1:
        System.out.print("Message 1");
        break;
    case 2:
        System.out.print("Message 2");
        break;
    case 3:
        Traitement();
        System.out.print("Message 3");
        break;
    default:
        System.err.print("Erreur !");
}
  
```

Exemple 3**Diagramme de séquence système : Fragment OPT (if)**

Traduction en Java

```

if(a == 1)
    System.out.print("Message 1");
    System.out.print("Message 2");
  
```

▪ Fragments : Les structures itératives

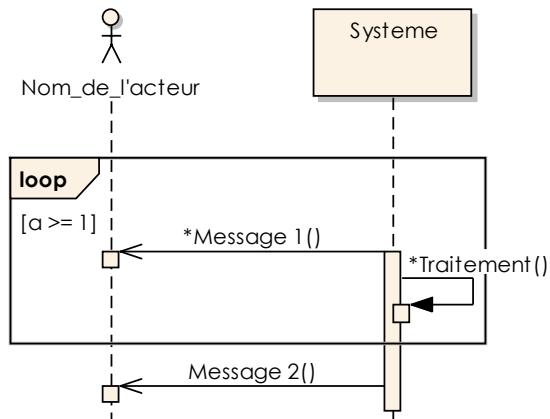
Définition : Structures itératives

Les structures itératives sont représentées en UML par le fragment **LOOP**

Exemple :

Diagramme de séquence système : Fragment LOOP

Diagramme de séquence



Traduction en Java

```

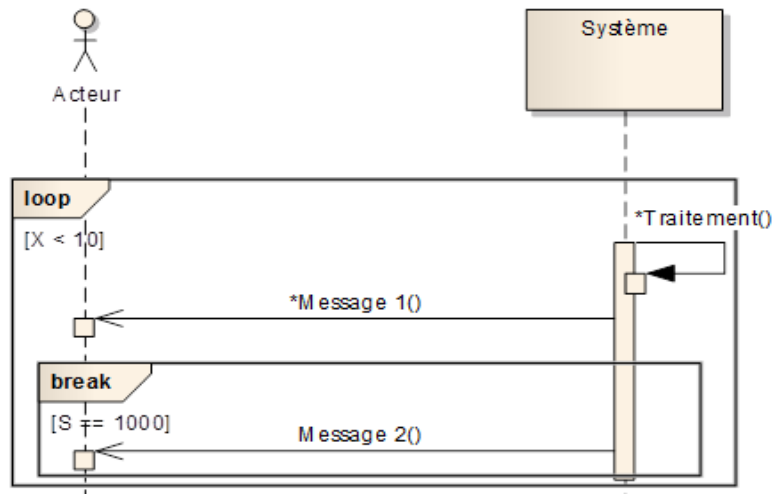
while (a >= 1){
    System.out.print("Message 1");
    Traitement();
}
System.out.print("Message 2");
  
```

Sortir d'une boucle :

Une structure LOOP est interrompue dans 2 cas :

- La condition d'itération n'est plus **TRUE**
- Utilisation du fragment **BREAK**

Diagramme de séquence



Traduction en Java

```

while (X < 10){
    Traitement();
    System.out.println("Message 1");
    if(S == 1000){
        System.out.print("Message 2");
        break;
    }
}
  
```



REMARQUE

L'interaction « **Message 2** » n'est pas itérative, car même si elle appartient à un fragment LOOP, elle n'est exécutée qu'une seule fois.

Exercice :

Guichet automatique bancaire (GAB)

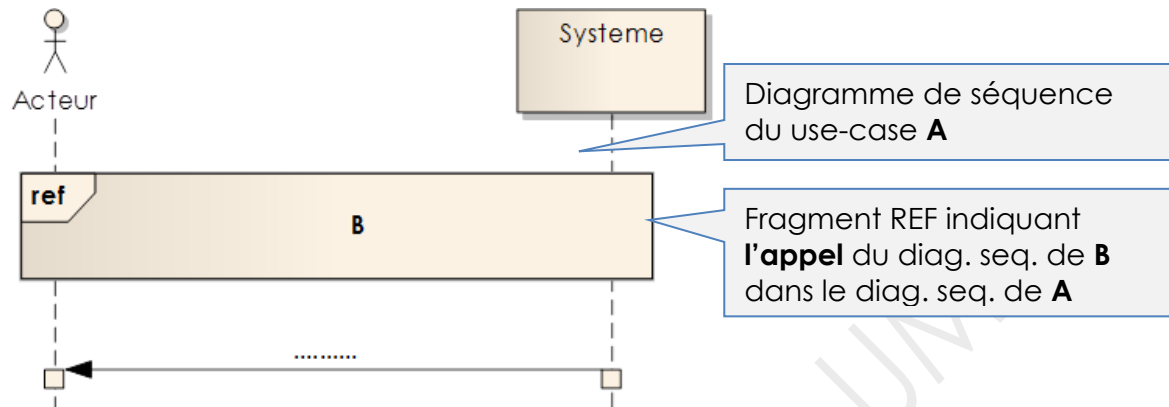
Travail à faire :

- Réaliser le diagramme de séquences du cas d'utilisation *Authentification* en utilisant les fragments. (Etudier les cas où la vérification du mot de passe se fait « une seule fois » et « 3 fois »)

▪ Fragments : Les références

Définition : Le fragment REF

Le fragment **REF** sert à modéliser – au niveau du diagramme de séquences – une relation d'inclusion ou d'extension entre deux cas d'utilisation **A** et **B** au niveau du UC-Diagram. Ce fragment évite aux concepteurs de réintroduire les messages de **B** dans le diagramme de séquence de **A**

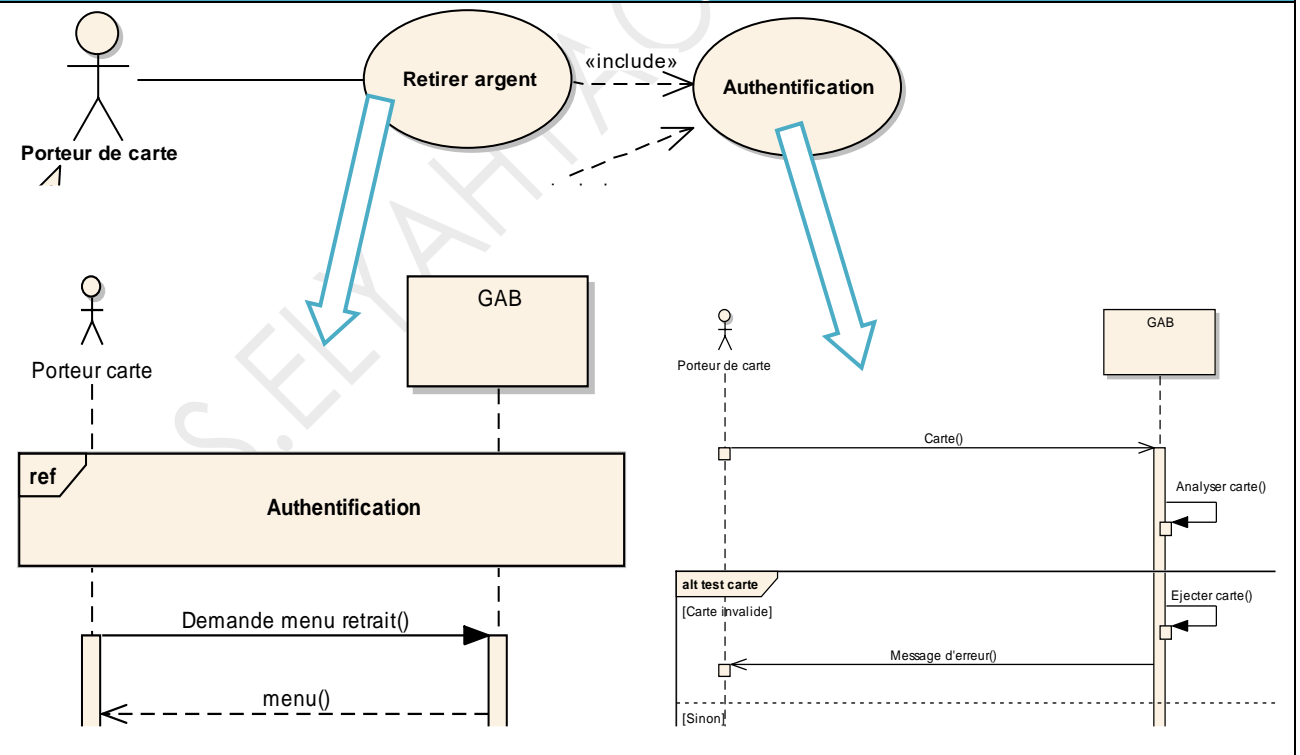


Exemple :

Diagramme de séquence système : Fragment REF

Etude de cas : G.A.B

UC. Authentification est inclus dans UC. Retirer argent



Etudes de cas :

Guichet automatique bancaire

[.....

N'importe quel porteur de carte bancaire peut retirer de l'argent, ou (s'il le souhaite) retirer de l'argent avec un reçu.

Si le porteur est un client de la banque, il doit pouvoir accéder aussi aux services suivants (en plus de celui du retrait d'argent) :

Déposer de l'argent, retirer un mini-relevé bancaire de son compte, consulter son solde, payer la facture d'abonnement du téléphone fixe et modifier ses paramètres de sécurité.

Un porteur de carte ne peut accéder à aucun service du GAB sans être d'abord authentifié.

.....]

Travail à faire :

- Identifier les acteurs et leurs cas d'utilisation associés
- Réaliser le diagramme des cas d'utilisation
- Réaliser le diagramme de séquence du use-case « Authentification »
- Rédiger les scénarios du use-case « Retirer argent » (considérer l'option du choix de retrait avec ou sans reçu)
- Réaliser le diagramme de séquence du use-case « Retirer argent »

Gestion commerciale

[.....

Pour passer une commande le client s'adresse au réceptionniste, celui-ci passe la commande (commande normale ou express) si le client existe déjà, sinon il doit d'abord le créer.

Le comptable ou le directeur des ventes peuvent enregistrer la facture de chaque commande dont le paiement est dûment réglé.

Le livreur doit enregistrer chaque livraison qu'il effectue.

Pour les commandes dont la liste de produits n'existe pas en quantité suffisante, le directeur des ventes peut aussi gérer le stock pour éviter toute rupture de produits.

La gestion du stock contient les fonctionnalités suivantes :

Gérer un produit, et enregistrer une demande d'alimentation du stock

La gestion d'un produit contient les fonctionnalités suivantes :

Ajouter, supprimer et mettre à jour un produit

Chaque utilisateur du système, doit d'abord s'authentifier avant de pouvoir accéder à n'importe quelle fonctionnalité.

.....]

Travail à faire :

- Identifier les acteurs et leurs cas d'utilisation associés
- Réaliser le diagramme des cas d'utilisation

- Réaliser le diagramme de séquence du cas « **Authentification** »
- Réaliser le diagramme de séquence du cas « **Ajouter client** »

[..... Lors de la réalisation du use-case « Passer commande », le système doit demander l'identifiant du client avant de demander le contenu de la commande (identifiants et quantités commandées de chaque produits). Si l'identifiant saisi n'existe pas, la fenêtre de la fonctionnalité « Ajouter client » doit automatiquement s'afficher, celle-ci doit d'abord terminer son traitement (ajout du client) avant de continuer le traitement de la fenêtre d'ajout de la commande.
.....]

- Réaliser le diagramme de séquence du cas « **Passer commande** » de l'acteur « **Réceptionniste** »

Gestion de médiathèque

[.....

Le système d'informations d'une médiathèque de livres informatiques comporte deux interfaces graphiques : un site internet pour les clients, et une application Windows pour les bibliothécaires et l'administrateur.

Le site internet est ouvert à tout internaute pour parcourir la liste des livres, afficher le sommaire d'un livre ou encore, créer un compte d'adhérent.

Si l'internaute est un adhérent, il peut aussi (après s'être authentifié) créer un panier, enregistrer un panier pour une modification ultérieure, ouvrir un panier déjà créé, valider l'achat d'un panier (un panier contient au moins un livre).

Il peut aussi (l'adhérent) modifier les informations de son compte (nom, prénom, ...) ou signaler au système qu'il a oublié son mot de passe.

Le bibliothécaire (utilisateur de l'interface Windows) doit pouvoir ajouter ou supprimer un livre, ajouter ou supprimer une catégorie de livres (développement, base de données, systèmes & réseaux, ...), ou encore afficher la liste des paniers validés, et la liste des paniers livrés.

Un livre ne doit pas exister dans la médiathèque, sans appartenir à une catégorie. Donc lors de l'ajout d'un livre, sa catégorie doit être spécifiée. S'il s'agit d'une nouvelle catégorie, elle doit être d'abord créée.

L'administrateur (lui aussi utilisateur de l'interface Windows) doit pouvoir afficher tous les comptes utilisateurs (qu'ils soient de type adhérent ou bibliothécaire), supprimer ou créer un compte bibliothécaire, ou modifier les informations de son propre compte à lui (nom, prénom, mot de passe, ...).

Il doit aussi (l'administrateur) pouvoir afficher la liste des paniers validés, et la liste des paniers livrés, et afficher la recette du jour et la recette du mois.

Bibliothécaires et administrateur doivent aussi s'authentifier.
.....]

Travail à faire :

- Identifier les acteurs et leurs cas d'utilisation associés

- Réaliser le diagramme des cas d'utilisation
- **Réaliser le diagramme de séquence du cas « Authentification »**
- **Réaliser le diagramme de séquence du cas « Créer catégorie livre »**
- **Réaliser le diagramme de séquence du cas « Ajouter livre »**

S.ELYAHYAOU - UML

C. Diagramme d'activité

1. Définition

Définition

Le **diagramme d'activité** est très utile pour décrire toute sorte de processus contenant des enchainements d'évènements, de flux de traitements et de flux de données.

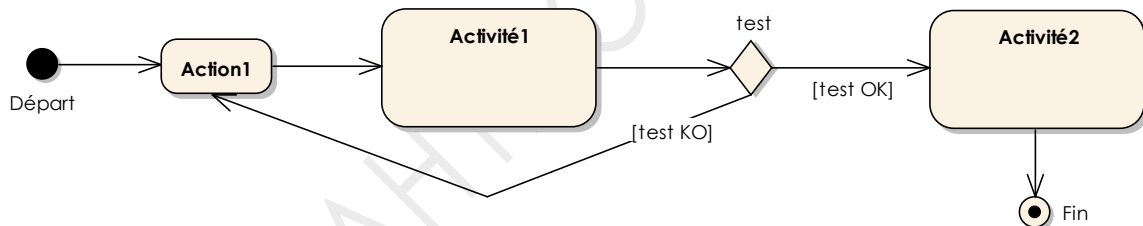
On peut donc l'utiliser pour représenter graphiquement :

- Le déroulement de l'activité globale d'un système, ou
- Le déroulement de l'un des processus métier exécutés par le système (Ex. : cheminement du traitement d'ajout d'un client), ou encore
- L'algorithme d'une fonction, ...

REMARQUE

Le diagramme d'activité peut être assimilé au model conceptuel ou organisationnel de traitements dans la méthode Merise

L'aspect graphique d'un DAC est comme ceci :



2. Réaliser le diagramme d'activité

a. Formalisme graphique

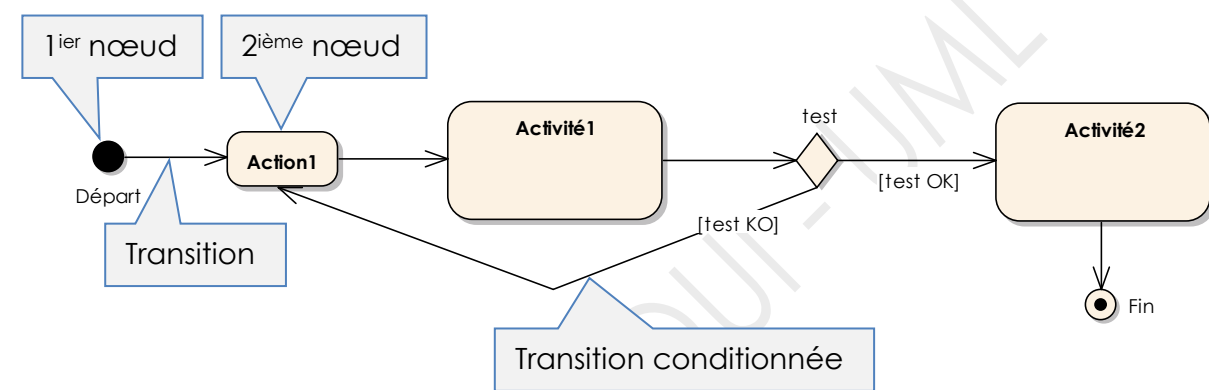
Le formalisme graphique du DAC se base sur deux principaux éléments : les « **nœuds** » et les « **transitions** ».

→ DAC : Les transitions (FLOW)

Définition

Une transition, appelée aussi « **flot** », est une association qui marque le passage d'un nœud vers un autre dans un DAC.

Comme les messages d'un DSEC, une transition peut être conditionnée par une condition de garde.

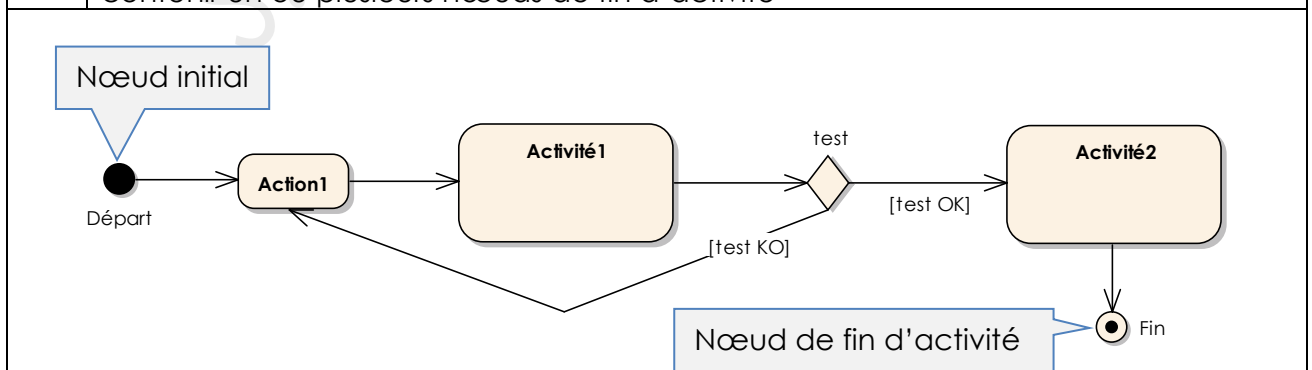


→ DAC : Les nœuds

▪ Nœud initial / Nœud final

Définitions

	Nœud initial Marque le début de tous les processus, un DAC doit contenir un seul nœud initial
	Nœud final Appelé aussi nœud de fin d'activité , marque la fin de tout le processus, un DAC doit contenir un ou plusieurs nœuds de fin d'activité



▪ Nœuds d'action

Définitions

Action :


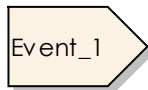
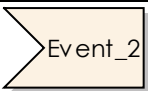
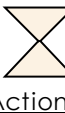
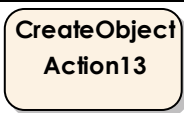
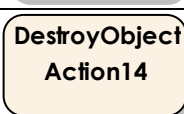
Une action est la plus petite unité ou traitement qu'on peut représenter dans un DAC, cela signifie que c'est un acte *atomique*, qui ne peut être divisible.

En termes de langage de programmation, on peut assimiler une action à une « instruction »

Exemples d'actions :

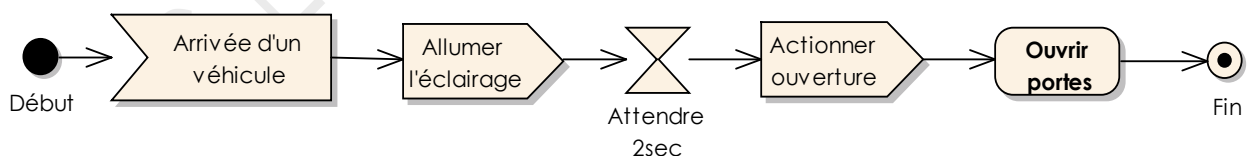
- Un appel de méthode
- Une affectation ou opération arithmétique
- La création / destruction d'un objet
- ...

Principaux types d'actions

	Action simple Opération ou traitement de base
	L'envoi d'un signal L'envoi d'un signal (ou événement) par le système
	La réception d'un signal La réception d'un signal (ou événement) par le système
	Un événement temporel Le système doit attendre un certain temps, avant de pouvoir passer au prochain nœud
	La création d'un objet Action qui a pour résultat la création d'un nouvel objet d'une classe Ex. l'appel d'un constructeur
	La destruction d'un objet Action qui a pour résultat la destruction d'un objet d'une classe Ex. l'appel d'un destructeur

Exemple : DAC – Nœuds d'actions

Etude de cas : Système de portail électronique



▪ Nœuds d'activité

Définition : ACTIVITE

Vérifier mot de passe

Comme on vient de voir, une action est une instruction élémentaire, mais une activité correspond plutôt à un traitement, composé de plusieurs instructions.

Une activité est donc **un enchaînement d'actions**, acheminé par des transitions et des **nœuds de contrôle** (Nœuds de contrôle : voir paragraphes suivants)

En termes de langage de programmation, on peut assimiler une activité au corps d'une méthode par ex.

Définition : ACTIVITE STRUCTUREE

«structured»
Supprimer client

Une activité structurée (on dit aussi **activité composite**) correspond à un traitement complexe, qu'on ne peut représenter dans un simple nœud d'activité, mais plutôt par un enchaînement **d'activités**, acheminé par des transitions et des nœuds de contrôle.

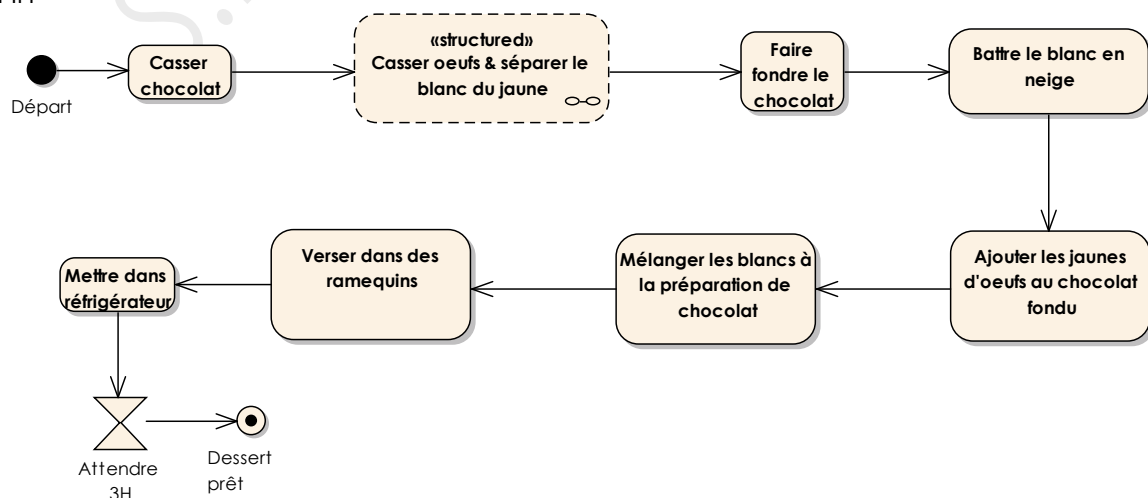
En termes de langage de programmation, on peut assimiler une activité composite au corps d'une méthode, qui lui-même contient l'appel d'une ou plusieurs méthodes.

Exemple : DAC – Nœuds d'actions & d'activité

Etude de cas : Recette de préparation du dessert : Mousse au chocolat (1)

Recette simplifiée :

- Casser les tablettes de chocolat
- Casser les œufs et séparer le blanc du jaune
- Faire fondre le chocolat
- Battre les blancs d'œufs en neige
- Ajouter les jaunes d'œufs au chocolat fondu
- Mélanger les blancs d'œufs battus à la préparation du chocolat (mixée avec les jaunes d'œufs)
- Verser le résultat dans des ramequins
- Mettre les ramequins au réfrigérateur
- Attendre 3 heures
- Fin



- **Nœuds de contrôle**
 - **Nœud de décision**

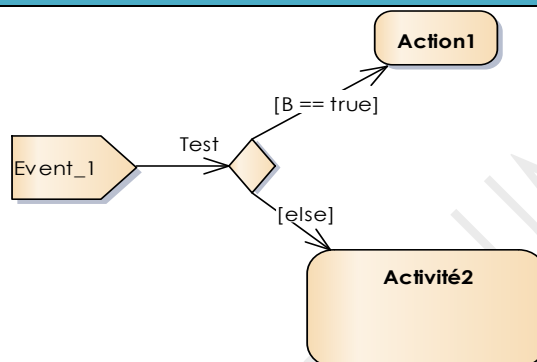
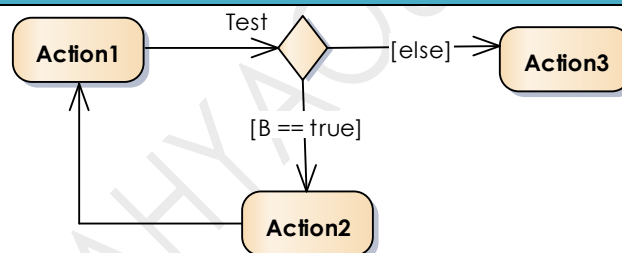
Définition

Test



Ce nœud permet de modéliser le test d'une condition booléenne. Un nœud de décision possède une transition entrante, et deux ou plusieurs transitions sortantes, ces dernières doivent être conditionnées.

En termes de langage de programmation, on peut assimiler ce nœud à une structure alternative.

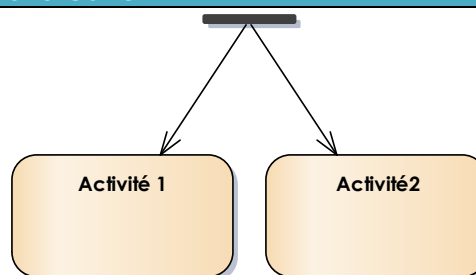
Exemple 1 : DAC – Nœud de décision**Exemple 2 : DAC – Nœud de décision**

- **Nœud de bifurcation / d'union (FORK / JOIN)**

Définition

Ce nœud permet de modéliser le lancement de plusieurs processus ou tâches simultanément. Il est aussi utilisé pour l'union ou la fusion de deux ou plusieurs transitions.

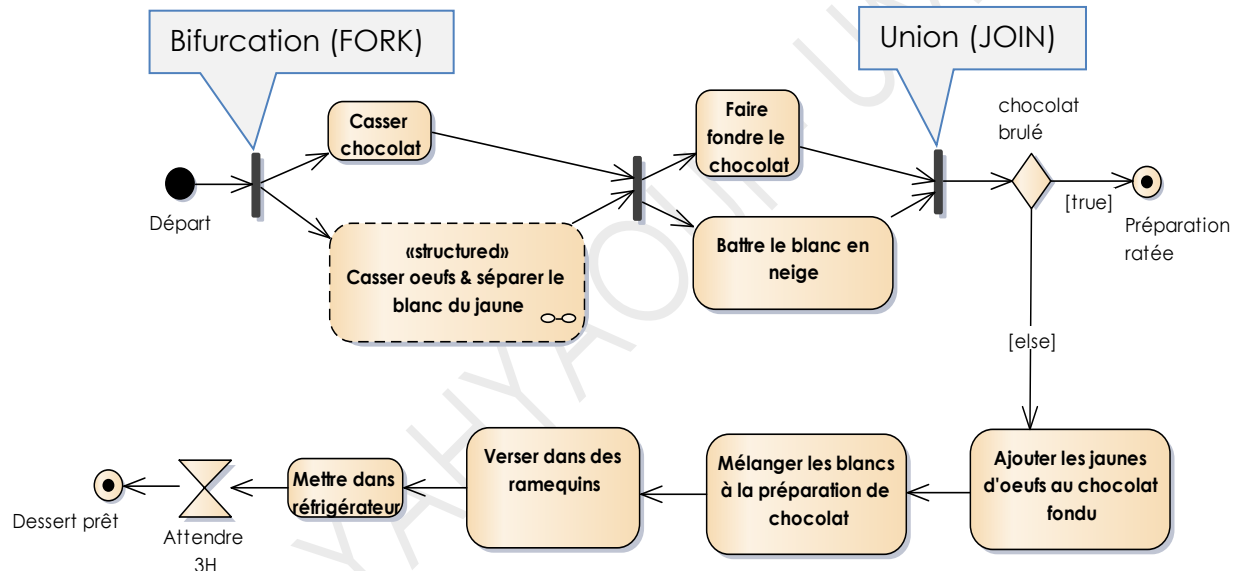
En termes de langage de programmation, ce nœud est particulièrement utile pour modéliser des situations d'utilisation de processus concurrents.

Exemple : DAC – Nœud de bifurcation

Exemple : DAC – Nœuds d'actions & d'activité**Etude de cas : Recette de préparation du dessert : Mousse au chocolat (2)**

Recette reformulée :

- Casser les tablettes de chocolat, et en parallèle, casser les œufs et séparer le blanc du jaune.
- Faire fondre le chocolat et en parallèle, battre les blancs d'œufs en neige.
- Si le chocolat est brûlé, on arrête la préparation, sinon, on ajoute les jaunes d'œufs au chocolat fondu.
- Mélanger les blancs d'œufs battus à la préparation du chocolat (mixée avec les jaunes d'œufs)
- Verser le résultat dans des ramequins
- Mettre les ramequins au réfrigérateur
- Attendre 3 heures
- Fin



▪ Nœuds d'objets

Définitions

Introduction :

Les flots qu'on a vus jusqu'à présent sont tous des transitions de **traitements**. Mais le DAC ne sert pas à modéliser uniquement les traitements, mais aussi les données, qui sont au cœur de tous les échanges orientés objets.

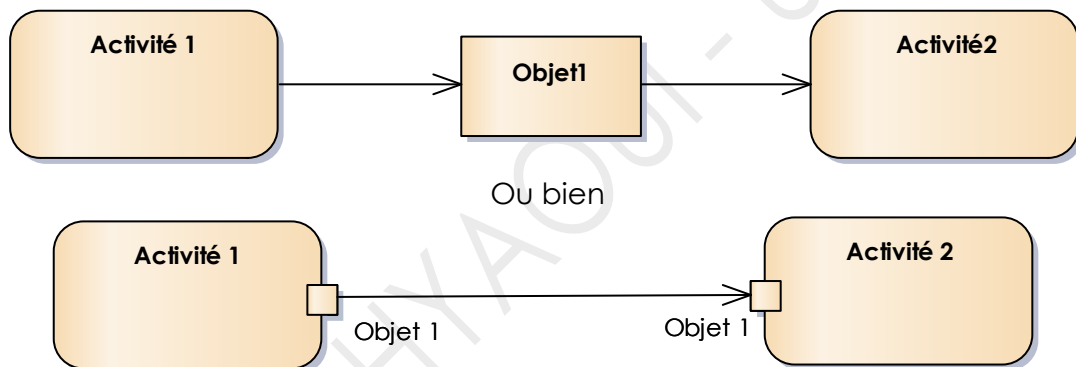
Nœud d'objet :

Un Nœud d'objet est la représentation d'une donnée transmise ou reçue par un flot (appelé **flot d'objet**)

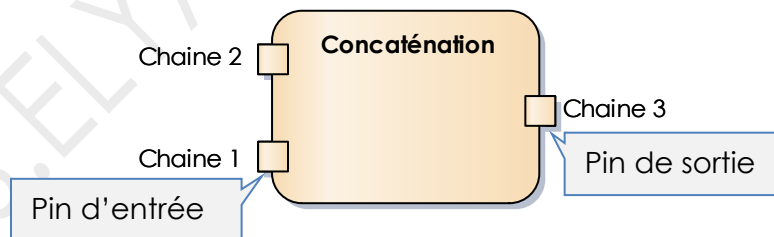
Exemples de flots d'objets :

- Passage de paramètres à une méthode
- Valeur retournée par une méthode
- Activité ou action qui se termine par la création / destruction d'un objet
- ...

Exemple 1 : DAC – Flots d'objets



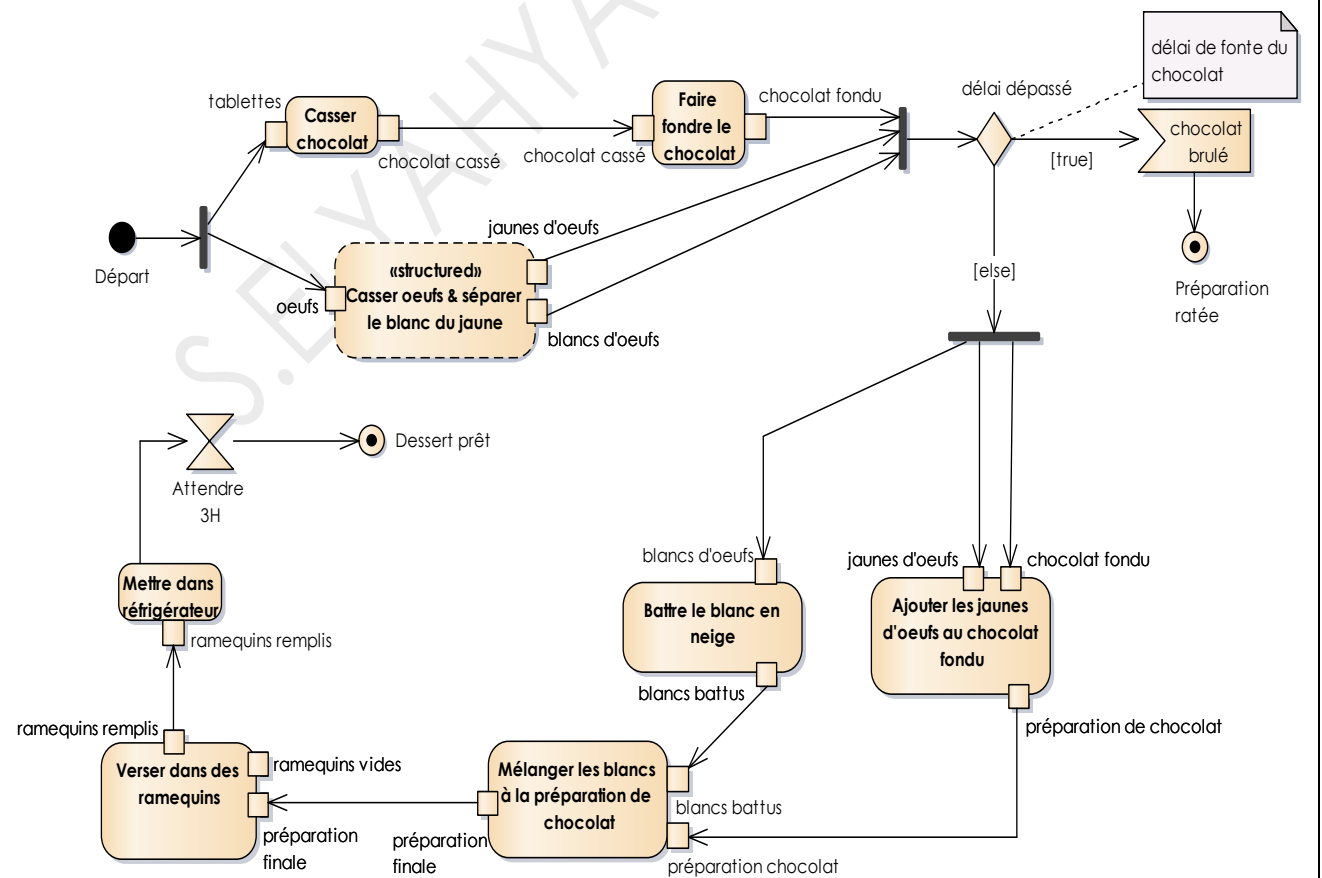
Exemple 2 : DAC – Flots d'objets



Exemple : DAC – Nœuds d'actions & d'activité**Etude de cas : Recette de préparation du dessert : Mousse au chocolat (3)**

Recette reformulée :

- Casser les tablettes de chocolat (**données d'entrée** : tablettes de chocolat / **données de sortie** : chocolat cassé),
et en parallèle,
Casser les œufs et séparer le blanc du jaune (**données d'entrée** : œufs / **données de sortie** : blanc d'œufs + jaunes d'œufs)
- Faire fondre le chocolat (**données d'entrée** : chocolat cassé / **données de sortie** : chocolat fondu)
et en parallèle,
Battre les blancs d'œufs en neige (**données d'entrée** : blancs œufs / **données de sortie** : blancs d'œufs battus)
- Si le chocolat est brûlé, on arrête la préparation,
sinon,
On ajoute les jaunes d'œufs au chocolat fondu (**données d'entrée** : chocolat fondu + jaunes d'œufs / **données de sortie** : préparation de chocolat)
- Mélanger les blancs d'œufs battus à la préparation du chocolat (**données d'entrée** : blancs d'œufs battus + préparation de chocolat / **données de sortie** : préparation finale)
- Verser le résultat dans des ramequins (**données d'entrée** : ramequins vides + préparation finale / **données de sortie** : ramequins remplis)
- Mettre les ramequins au réfrigérateur (**données d'entrée** : ramequins remplis / **données de sortie** : ramequins remplis)
- Attendre 3 heures
- Fin



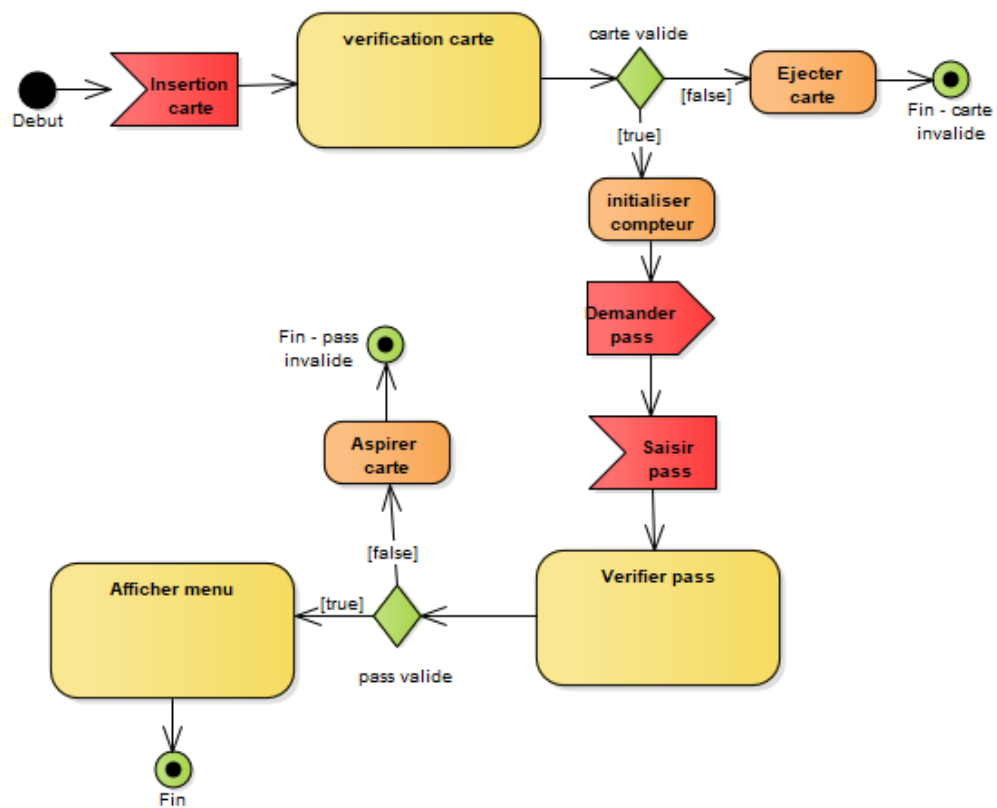
b. Diagramme d'activité « partiel »**Définition**

Un DAC est dit : « **partiel** » quand il sert à modéliser la réalisation de l'un des use-case du système.

Exemple 1 :

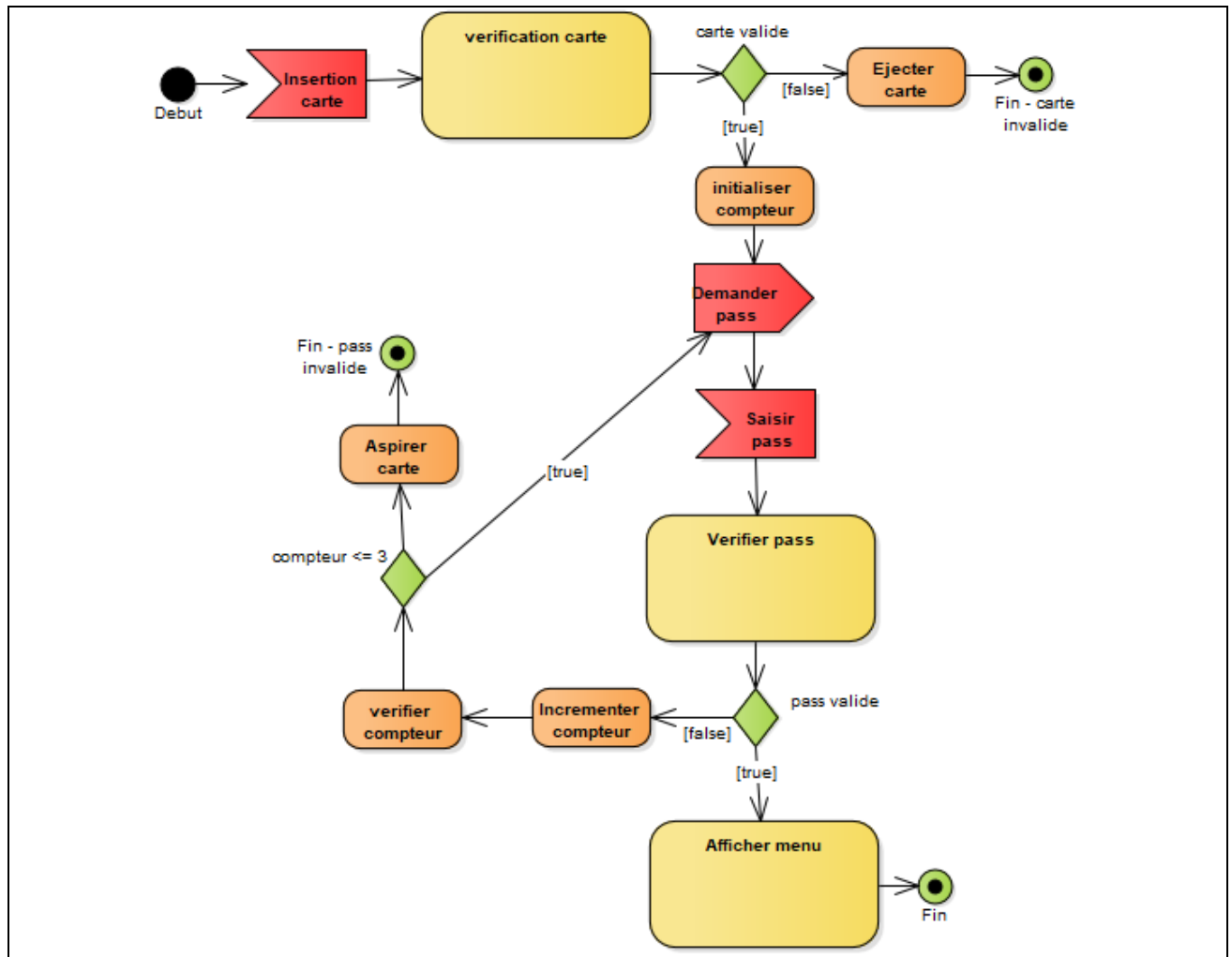
Etude de cas : GAB

Réaliser le DAC du use-case : Authentification (vérification mot de passe : 1 fois)

**Exemple 2 :**

Etude de cas : GAB

Réaliser le DAC du use-case : Authentification (vérification mot de passe : 3 fois)



c. Diagramme d'activité « global »

Définition

Un DAC est dit : « **global** » quand il sert à modéliser l'activité globale de tout le système. Souvent dans un DAC global, chaque activité représente un cas d'utilisation du système.

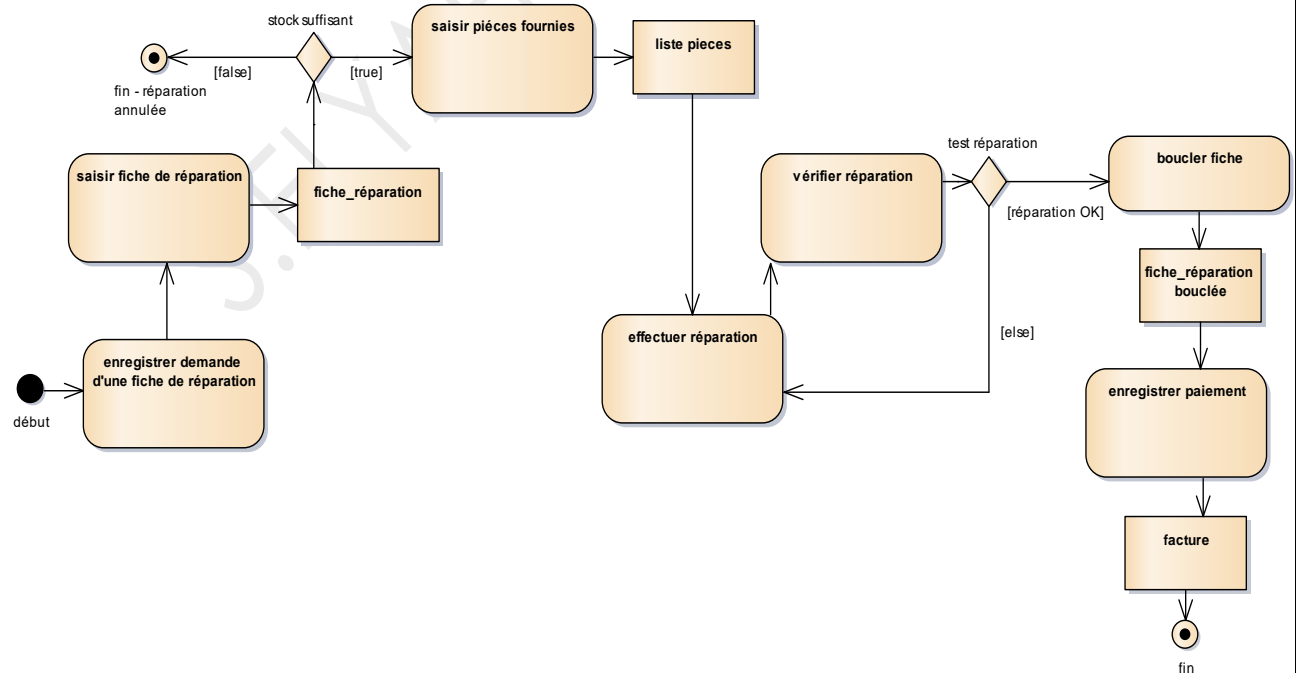
Exemple 1 :

Etude de cas : Atelier de services après-vente automobile

Réaliser le DAC global du processus de réparation

Le système informatique d'un atelier de réparation automobile offre plusieurs fonctionnalités (use-cases). Le processus de déroulement de toutes ses fonctionnalités est décrit dans les spécifications suivantes :

- Un mécanicien enregistre la demande d'une fiche de réparation
- Quand la demande est enregistrée, l'un des chefs-mécaniciens doit pouvoir saisir la fiche de réparation
- Cette fiche de réparation servira de base pour le magasinier pour l'opération de saisie de la liste des pièces nécessaires à la réparation.
- Si le magasin ne contient pas les pièces demandées, le processus est annulé, sinon la liste des pièces est générée.
- Quand la liste des pièces est fournie, le mécanicien peut effectuer la réparation et l'enregistrer dans le système.
- Quand la réparation est enregistrée, le chef mécanicien doit la vérifier (par rapport à la fiche de réparation) pour voir si la réparation a été bien exécutée. Si c'est le cas, il doit effectuer l'opération de bouclage de la fiche de réparation, cette opération donnera la création d'une fiche de réparation bouclée. Sinon, on retourne à l'opération 5.
- Quand la fiche bouclée est créée, le comptable doit pouvoir enregistrer le paiement réglé par le client, cette opération se terminera par la génération d'une facture.
- Fin

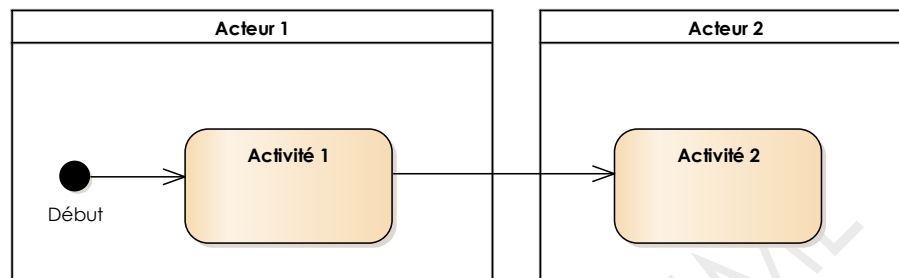


REMARQUE

Dans cette solution, les activités sont représentées sans mentionner l'acteur auquel appartient chaque activité.

Définition : « Swimlanes »

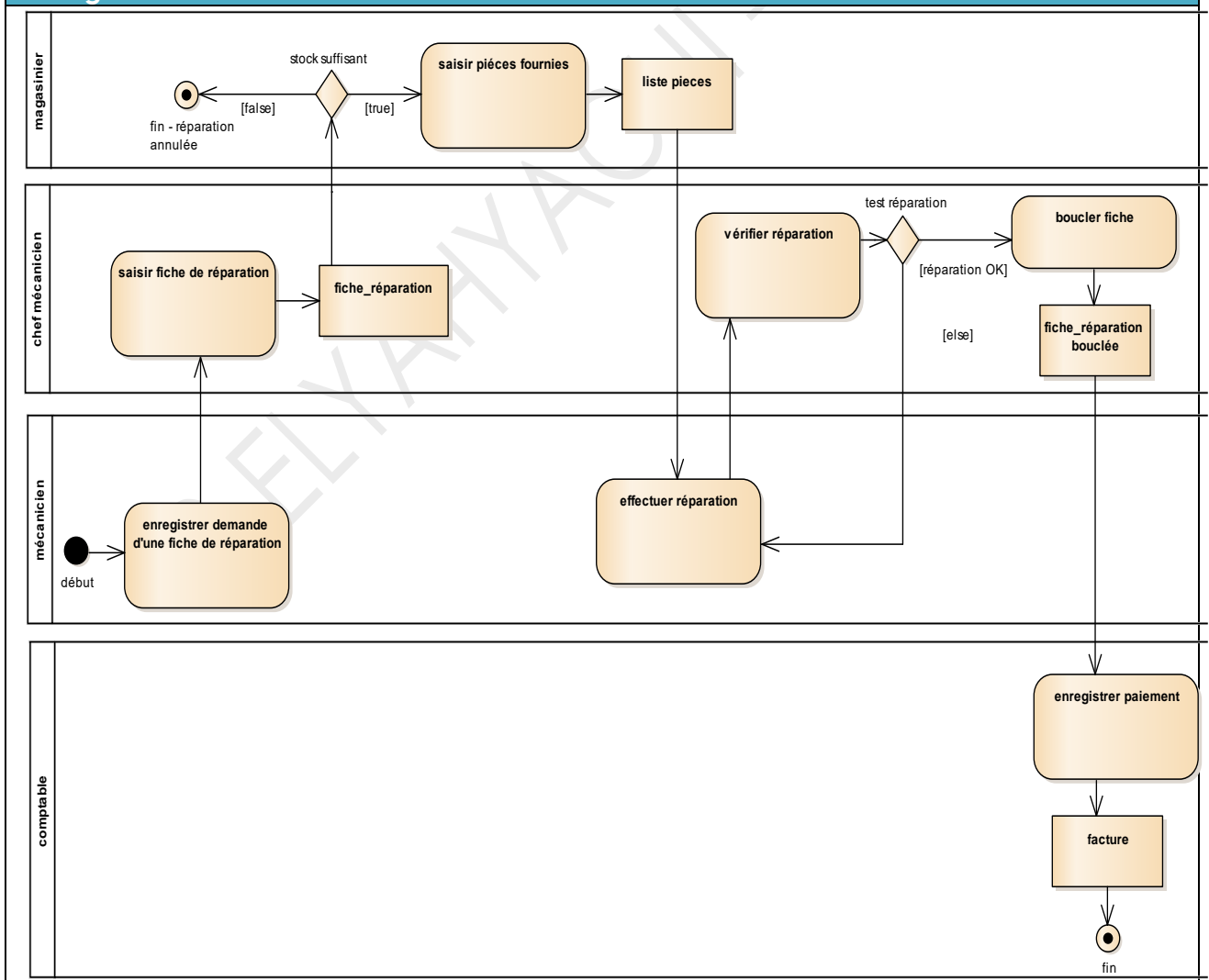
Un **swimlane**, ou **ligne d'eau**, ou **partition**, représente un couloir qui regroupe les nœuds d'activité appartenant à chaque acteur du système.



Exemple 2 :

Etude de cas : Atelier de services après-vente automobile

DAC global : 2^{ème} solution



d. Etudes de cas**Guichet automatique bancaire**

[.....

N'importe quel porteur de carte bancaire peut retirer de l'argent, ou (s'il le souhaite) retirer de l'argent avec un reçu.

Si le porteur est un client de la banque, il doit pouvoir accéder aussi aux services suivants (en plus de celui du retrait d'argent) :

Déposer de l'argent, retirer un mini-relevé bancaire de son compte, consulter son solde, payer la facture d'abonnement du téléphone fixe et modifier ses paramètres de sécurité.

Un porteur de carte ne peut accéder à aucun service du GAB sans être d'abord authentifié.

.....]

Travail à faire :

- Réaliser le diagramme d'activité du use-case « Authentification »
- Réaliser le diagramme d'activité du use-case « Retirer argent »

Gestion commerciale

[.....

Pour passer une commande le client s'adresse au réceptionniste, celui-ci passe la commande (commande normale ou express) si le client existe déjà, sinon il doit d'abord le créer.

Le comptable ou le directeur des ventes peuvent enregistrer la facture de chaque commande dont le paiement est dûment réglé.

Le livreur doit enregistrer chaque livraison qu'il effectue.

Pour les commandes dont la liste de produits n'existe pas en quantité suffisante, le directeur des ventes peut aussi gérer le stock pour éviter toute rupture de produits.

La gestion du stock contient les fonctionnalités suivantes :

Gérer un produit, et enregistrer une demande d'alimentation du stock

La gestion d'un produit contient les fonctionnalités suivantes :

Ajouter, supprimer et mettre à jour un produit

Chaque utilisateur du système, doit d'abord s'authentifier avant de pouvoir accéder à n'importe quelle fonctionnalité.

.....]

Travail à faire :

- Réaliser le diagramme d'activité du cas « Authentification »
- Réaliser le diagramme d'activité du cas « Ajouter client »

[..... Lors de la réalisation du use-case « Passer commande », le système doit demander l'identifiant du client avant de demander le contenu de la commande (identifiants et quantités commandées de chaque produits). Si l'identifiant saisi n'existe pas, la fenêtre de

la fonctionnalité « Ajouter client » doit automatiquement s'afficher, celle-ci doit d'abord terminer son traitement (ajout du client) avant de continuer le traitement de la fenêtre d'ajout de la commande.

.....]

- **Réaliser le diagramme d'activité du cas « Passer commande » de l'acteur « Réceptionniste »**

Gestion de médiathèque

[.....

Le système d'informations d'une médiathèque de livres informatiques comporte deux interfaces graphiques : un site internet pour les clients, et une application Windows pour les bibliothécaires et l'administrateur.

Le site internet est ouvert à tout internaute pour parcourir la liste des livres, afficher le sommaire d'un livre ou encore, créer un compte d'adhérent.

Si l'internaute est un adhérent, il peut aussi (après s'être authentifié) créer un panier, enregistrer un panier pour une modification ultérieure, ouvrir un panier déjà créé, valider l'achat d'un panier (un panier contient au moins un livre).

Il peut aussi (l'adhérent) modifier les informations de son compte (nom, prénom, ...) ou signaler au système qu'il a oublié son mot de passe.

Le bibliothécaire (utilisateur de l'interface Windows) doit pouvoir ajouter ou supprimer un livre, ajouter ou supprimer une catégorie de livres (développement, base de données, systèmes & réseaux, ...), ou encore afficher la liste des paniers validés, et la liste des paniers livrés.

Un livre ne doit pas exister dans la médiathèque, sans appartenir à une catégorie. Donc lors de l'ajout d'un livre, sa catégorie doit être spécifiée. S'il s'agit d'une nouvelle catégorie, elle doit être d'abord créée.

L'administrateur (lui aussi utilisateur de l'interface Windows) doit pouvoir afficher tous les comptes utilisateurs (qu'ils soient de type adhérent ou bibliothécaire), supprimer ou créer un compte bibliothécaire, ou modifier les informations de son propre compte à lui (nom, prénom, mot de passe, ...).

Il doit aussi (l'administrateur) pouvoir afficher la liste des paniers validés, et la liste des paniers livrés, et afficher la recette du jour et la recette du mois.

Bibliothécaires et administrateur doivent aussi s'authentifier.

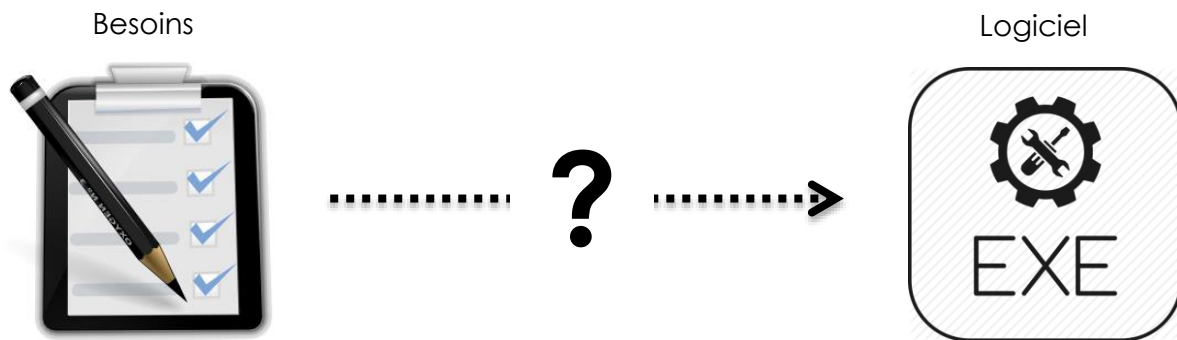
.....]

Travail à faire :

- **Réaliser le diagramme d'activité du cas « Authentification »**
- **Réaliser le diagramme d'activité du cas « Créer catégorie livre »**
- **Réaliser le diagramme d'activité du cas « Ajouter livre »**

D. Résumé : Comment mettre en œuvre UML

a. Problématique



- UML est, rappelons-le, un langage graphique, qui ne sert qu'à faire des modèles, il n'offre aucune méthodologie ou démarche pour encadrer et piloter les étapes de conception & réalisation du futur système informatique.

Une méthode de développement (Exemples : **Unified Process**, **Extreme Programming**, ...) fourni justement un plan de travail bien défini, qui décrit le déroulement de toutes les étapes de modélisation et réalisation du projet, depuis la définition des besoins, jusqu'à la livraison du produit final, tout en respectant le délai prévu.

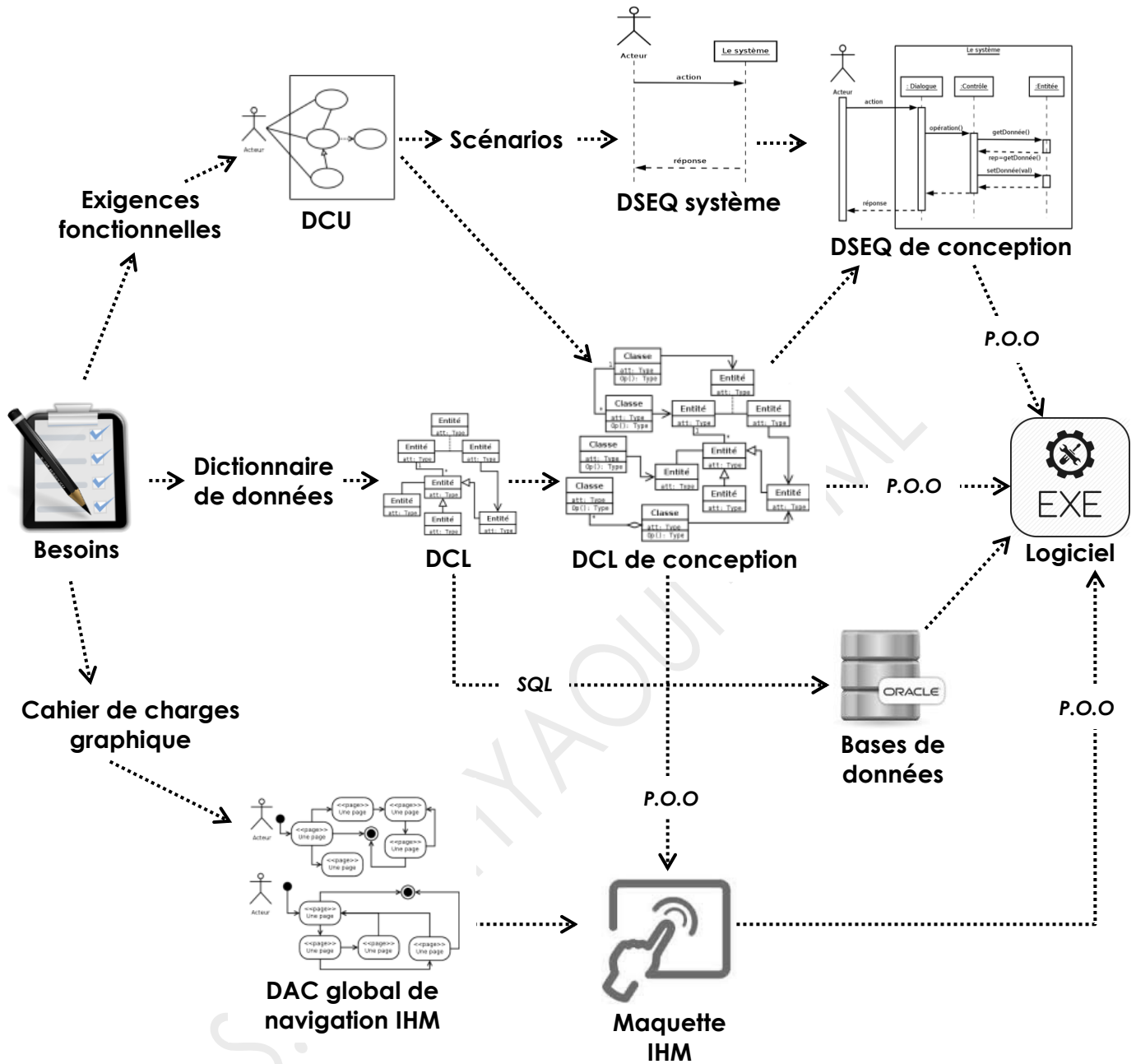
- Les diagrammes UML ne sont pas tous nécessaires pour chaque projet. En fait, rien n'impose à l'analyste le choix de tel ou tel diagramme, sauf bien sûr, certains diagrammes indispensables comme celui des **cas d'utilisation** ou celui des **classes**.

Le choix se fait selon la nécessité de modéliser tel ou tel aspect du système (déploiement & architecture logicielle, activité globale et navigabilité des interfaces graphiques, ...), et aussi selon les préférences techniques de l'analyste (Exemple : modéliser l'algorithme d'un traitement ou d'une fonction avec un DAC au lieu d'un DSEQ)

b. Plan de travail proposé

Trois axes :

- **Le fonctionnel** : Les comportements du logiciel. Centré sur le **DCU**
- **La base de données** : Les données que le système doit gérer et stocker. Centré sur le diagramme des classes **DCL**
- **L'IHM (Interfaces Homme-Machine)** : L'ensemble des interfaces graphiques avec lequel dialoguera l'acteur. Centré sur le **cahier de charges graphique** et le **DAC de navigation** (DAC global qui modélise les cheminements possibles entre les différentes interfaces graphiques de tout le système)



REMARQUES

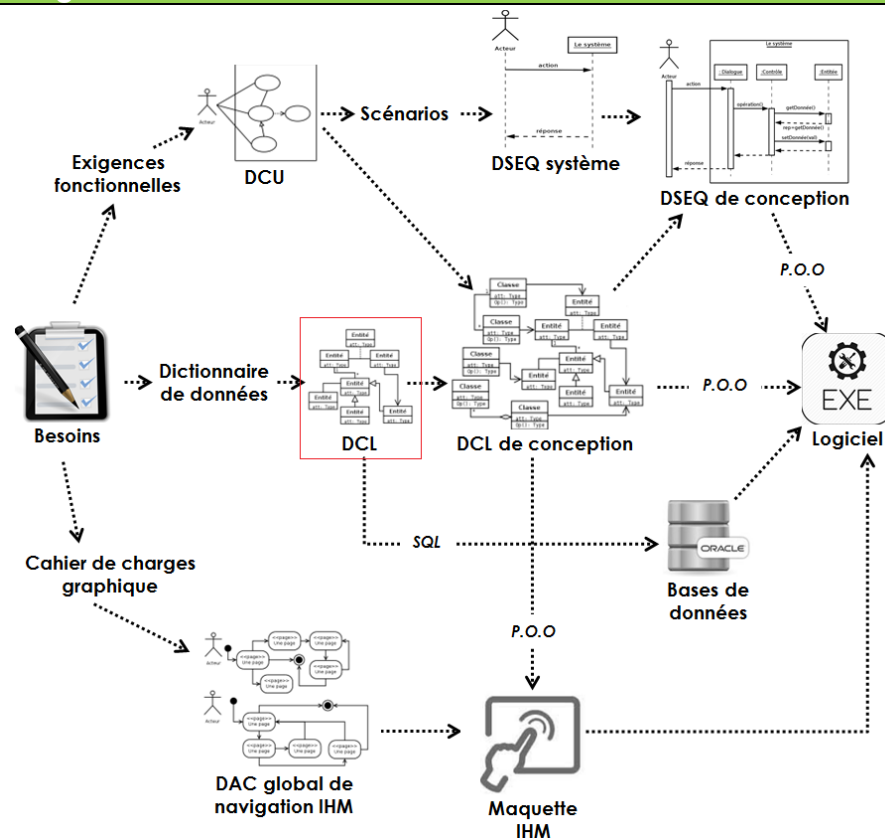
- Tout plan de travail peut contenir des défauts.
- Un plan de travail qui conviendrait parfaitement à un projet, peut ne pas convenir à un autre : les solutions "miracles" n'existent pas !
- Un bon plan de travail n'est pas suffisant, la réussite d'un projet dépend avant tout de la qualité de l'équipe qui le réalise.

III. Conception de la solution

A. Diagramme des classes « Modèle »

1. Définition

Diagramme de classes



Le **diagramme de classes** décrit les données que le système doit gérer, et la manière dont ces données seront organisées, à l'aide de « classes », qui seront transformées en tables lors du passage vers le modèle physique.

Plus concrètement, ces classes – appelées parfois « **les classes métier** » – vont regrouper toute la logique métier du système.

Exemples :

Si votre système gère des transactions commerciales, les classes métier seront donc : client, produit, commande, facture, ...

Si votre système gère les inscriptions dans un institut de formation, les classes métier seront donc : matière, enseignant, étudiant, ...



IMPORTANT

Le diagramme de classes représente l'une des étapes les plus critiques dans la réalisation de tout le système, vu qu'il représente la structure initiale du schéma de la future base de données.

(Les règles de passage du diagramme de classes vers les tables SQL seront détaillées dans le chapitre : De UML à SQL)

Le formalisme graphique du DCL repose sur les éléments suivants : les **classes** et les **liens d'association**.

2. Composants du diagramme de classes

a. Classes

Définition : Notion de « Classe »

Une **classe** est une famille d'individus qui possèdent tous (**sans exception**) :
Les mêmes **propriétés**, les mêmes **comportements** et la même sémantique.

Exemple

Prenons l'exemple des 3 voitures suivantes :

Voiture 1

Propriétés

Matricule : 1A22305
Marque : Renault
Modèle : Scénic
M.E.C : 15-01-2009
Energie : Diesel
Couleur : Gris
Prix : 156000

Comportements

Démarrer, freiner,
accélérer, ...

Voiture 2

Propriétés

Matricule : 5DF9930
Marque : Mercedes
Modèle : C220
M.E.C : 01-01-2008
Energie : Essence
Couleur : Noir
Prix : 350000

Comportements

Démarrer, freiner,
accélérer, ...

Voiture 3

Propriétés

Matricule : 2B10050
Marque : Renault
Modèle : Fluence
M.E.C : 20-02-2012
Energie : Essence
Couleur : Noir
Prix : 225000

Comportements

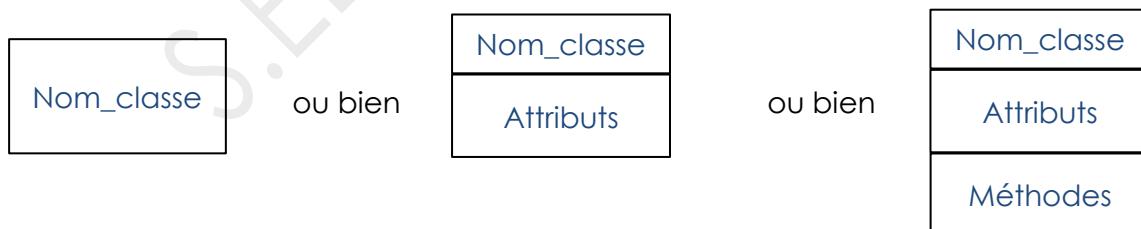
Démarrer, freiner,
accélérer, ...

On voit donc que n'importe quelle voiture appartient à la même catégorie, ou unité sémantique, possède les mêmes **propriétés** et les mêmes **comportements**.

On parle donc de la **classe** Voiture.

En UML :

- Les propriétés d'une classe sont appelées « **Attributs** ».
- Les comportements d'une classe sont appelées « **Méthodes** » ou « **Opérations** » (En P.O.O. les méthodes sont l'équivalent des fonctions en programmation structurée).
- Chaque individu d'une classe est appelé « **Objet** » ou « **Instance** ».
- Chaque objet possède les mêmes attributs et méthodes que ceux des autres objets de la même classe.
- Les classes sont notées graphiquement de 3 manières :



Définition : L'encapsulation

Généralement, les classes doivent collaborer pour réaliser un traitement.

Concrètement cela veut dire qu'une classe **A** peut avoir besoin d'accéder à un attribut ou une méthode d'une autre classe **B**.



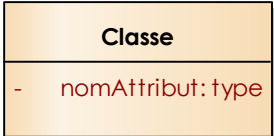
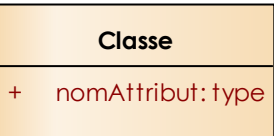
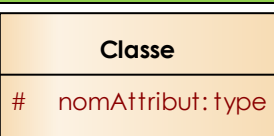
REMARQUE

Accéder à un membre (attribut ou méthode) d'une classe **A** depuis une classe **B**, signifie que : l'une des méthodes de **B** accède (en lecture ou en écriture) à un membre de **A**.

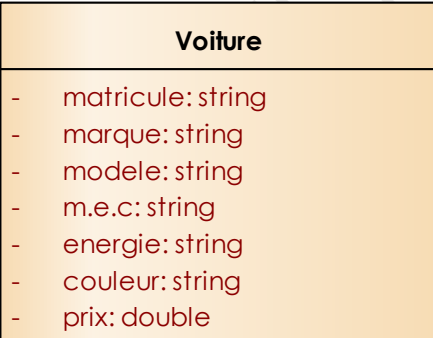

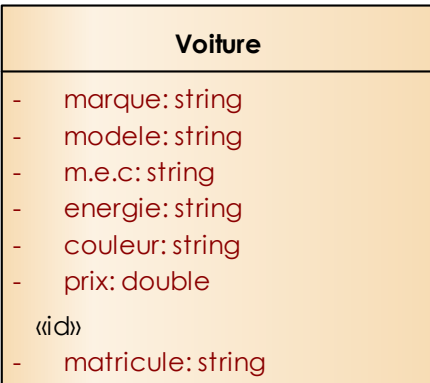
Dans ce contexte, chaque classe doit pouvoir désigner les membres qu'elle souhaite rendre publics pour les autres classes, et les membres qu'elle souhaite garder privés. C'est le principe **d'encapsulation**.

UML définit **3** niveaux d'accès pour accéder aux membres d'une classe depuis une autre classe :

- **Private**
- **Public**
- **Protected**

Private	
Notation UML : -	Définition
	C'est le niveau d'accès le <u>plus</u> restreint, aucune autre classe ne peut accéder au membre.
Public	
Notation UML : +	Définition
	C'est le niveau d'accès le <u>moins</u> restreint, toute autre classe peut accéder au membre.
Protected	
Notation UML : #	Définition
	Un niveau d'accès <u>intermédiaire</u> . Le membre est accessible seulement aux classes du même package , et aux classes « filles » (héritage). (Les notions d' héritage et de packages seront détaillées dans les prochains paragraphes)

Exemple : Classe Voiture

	
<div>  REMARQUE </div> <p>Pour désigner l'identifiant de la classe, on utilise le stéréotype « id »</p>	
	



IMPORTANT

Le principe d'encapsulation n'est pas implémenté par tous les langages orientés objet de la même manière.

Exemples :

- Le niveau d'accès par défaut pour **C++** est « **private** », tandis que pour **Java** c'est

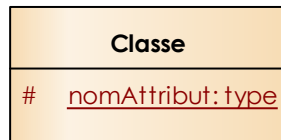
« **package** » (accessible seulement aux classes du même package).

- En Java, si on change de package, les membres **protected** ne sont pas **visibles**, **même pour les classes filles**.
- En C++, il est possible pour une méthode ou une classe d'accéder à un attribut **privé** d'une autre classe, en déclarant cette méthode ou classe comme étant « **amie** ». En Java, ce principe n'existe pas, il n'y aucun moyen d'accéder à un membre privé depuis l'extérieur.

Définition : Les membres « statiques » d'une classe

Un membre est dit « statique », si sa valeur (ou son comportement, s'il s'agit d'une méthode) est la même, pour tout individu de la classe.

Les attributs et méthodes statiques sont toujours **soulignés**.



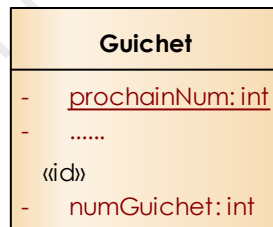
Les attributs (ou méthodes) statiques sont souvent appelés « attributs de classe », contrairement aux attributs non-statiques qui sont appelés « attributs d'objet ».

Exemple : Classe Guichet

Dans une file d'attente ordonnée par des numéros de tickets, comme dans les agences de paiement de factures, chaque guichet de paiement possède un afficheur. L'information « **prochain numéro** » doit être la même sur tous les afficheurs, indépendamment du guichet auquel appartient l'afficheur.

Donc pour le concepteur du programme, cela impose que si la classe **Guichet** doit contenir un attribut nommé par ex. « **prochainNum** », cet attribut doit avoir **la même valeur, quel que soit l'objet Guichet à partir duquel on souhaite le consulter**.

C'est donc un attribut statique.

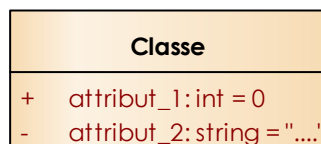


Définition : Attributs avec valeur par défaut

Les valeurs par défaut sont utilisées lorsqu'il est nécessaire qu'un attribut ait une valeur bien définie **dès l'instant de création de l'objet**.

Syntaxe :

Niveau ATTRIBUT : TYPE = VALEUR_PAR_DEFAUT



b. Liens d'associations**Définition : Association**

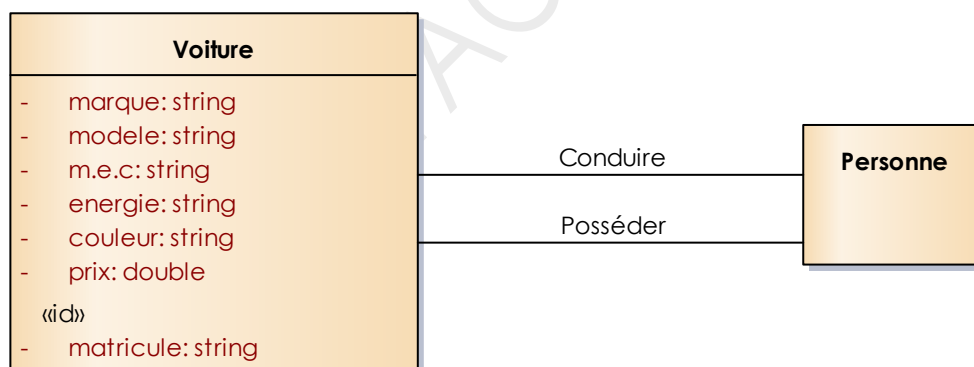
Une **association** est une relation qui existe entre une classe et elle-même, ou entre plusieurs classes.

Une association doit exister entre des classes, quand les objets de ces classes sont obligés d'avoir des liens entre eux.

**i REMARQUES**

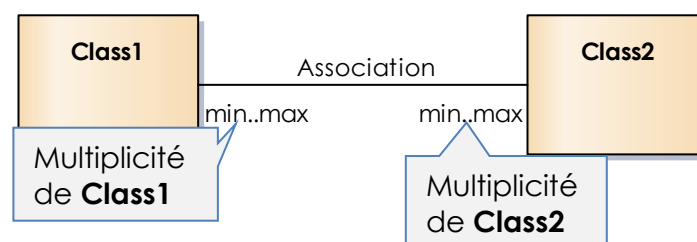
- Dans un diagramme de classe, chaque association doit avoir un nom unique (généralement un verbe)
- Une association entre une classe et elle-même est appelée *réflexive*
- Une association entre 2 classes est appelée association *binaires*
- Une association entre 3 classes est appelée association *ternaire*
- Une association entre 4 classes est appelée association *quaternaire*
- Les associations qui dépassent 2 classes sont fortement déconseillées
- Il est possible d'avoir plusieurs associations entre les mêmes classes.







Exemple :

**Définition : Multiplicités**

Le nombre (min et max) d'objets avec lesquels participe une classe dans une association, est appelé « **multiplicité** ».

A l'inverse de la notation Merise, la multiplicité est placée près de la classe ciblée.

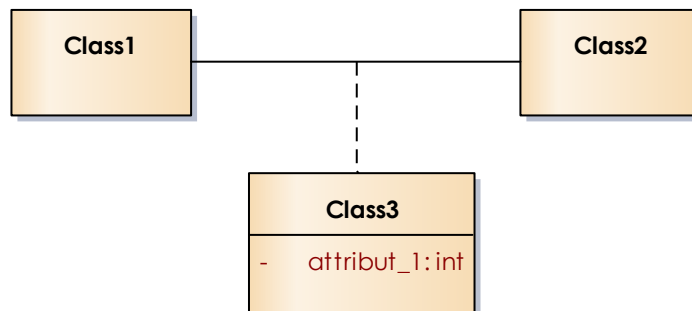


Notation des multiplicités	Correspondances Java
<p>« 0..1 » :</p>  <p>Un objet Class1 peut être lié à 0 ou un seul objet Class2.</p> <p>« 1..1 » ou bien « 1 » :</p>  <p>Un objet Class1 peut être lié à un et un seul objet Class2.</p>	<p>Correspondances Java</p> <pre>public class Class1 { Class2 x; }</pre>
<p>« 0..* » ou bien « * » :</p>  <p>Un objet Class1 peut être lié à 0 ou plusieurs objets Class2.</p> <p>« 1..* » :</p>  <p>Un objet Class1 peut être lié à 1 ou plusieurs objets Class2.</p> <p>« min..max » :</p>  <p>Un objet Class1 peut être lié à 2 ou 4 objets Class2.</p>	<pre>public class Class1 { Class2 x[]; }</pre>
<p>i REMARQUE</p> <ul style="list-style-type: none"> Quand min == max, on note uniquement l'un des deux. Pour des raisons de cohérence de la base de données, une association plusieurs à plusieurs, par ex. :  <p>doit obligatoirement avoir un nom. (Voir chap. De UML à SQL)</p>	

Définition : Classes d'association

Une classe d'association est un type particulier d'association, qui n'est utilisé **que quand un attribut appartient à l'union des classes impliquées dans l'association**.

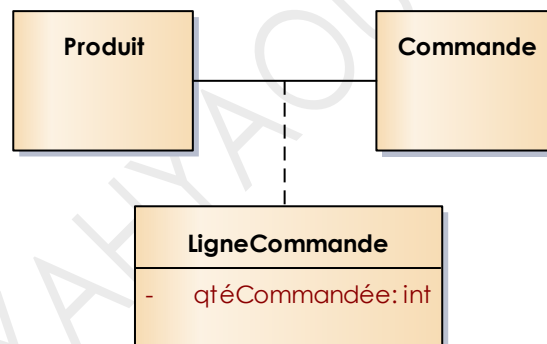
Dans ce cas, **c'est l'association qui contient l'attribut**, et non les classes liées.

**Exemple : Attribut d'association****Etude de cas : Gestion de commandes de clients**

L'un des exemples types de cette situation est le cas de l'association **Produit-Commande**.

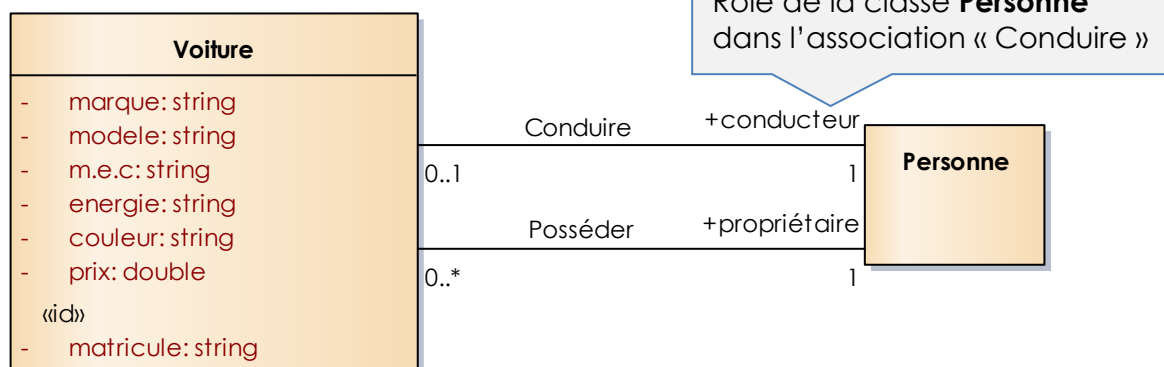
L'attribut **qtéCommandée** n'appartient ni à la classe **Produit**, ni à la classe **Commande**. Même plus que cela, il ne peut pas exister sans une association entre un objet **Produit** et un objet **Commande**.

C'est donc un attribut de la classe d'association **Produit-Commande** (Communément appelée « ligne de commande »).

**Définition : Rôles**

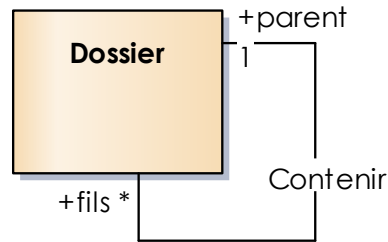
Dans une association, le « rôle » est un nom qui qualifie le rôle que joue la classe dans l'association.

Le rôle est noté avec le signe **+** prêt de la multiplicité.

Exemple 1 : Rôles**Exemple 2 : Importance des « Rôles » dans les associations réflexives**

Un dossier contient 0 ou plusieurs dossiers.

Un dossier est contenu dans un et un seul dossier.



IMPORTANT

Les rôles sont **obligatoires** quand il s'agit d'une association **réflexive**, ou une double (ou triple, ...) association entre **les mêmes classes**.

Définitions : Contraintes sur les associations

Généralement en UML, une « **contrainte** » est une condition imposée à un élément (association, classe, objet, use-case, attribut, ...) appartenant à un diagramme. Les contraintes sont souvent issues des exigences fonctionnelles ou techniques.

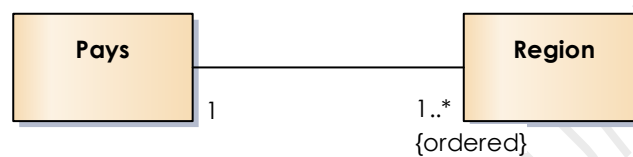
Pour les associations, une contrainte peut concerner l'**association elle-même**, ou l'**une des classes qui participent à l'association**.

- **Contrainte d'ordre (Tri d'objets)**

Signifie que, pour les classes ayant la multiplicité « **plusieurs** », l'ensemble des objets de la classe doit être trié selon un ordre précis.

Exemple :

Un pays contient plusieurs régions, qui doivent être triées par ordre alphabétique.

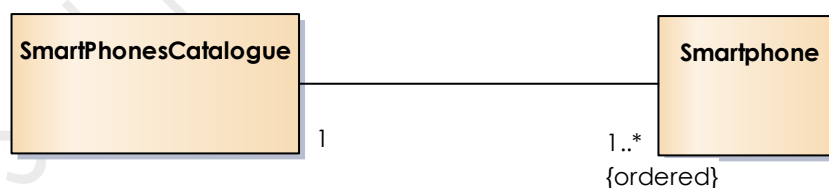


i REMARQUES

- Pour permettre les doublons, on utilise la contrainte {bag} (collection non-ordonnée permettant les doublons)
- Pour permettre les doublons, avec la contrainte d'ordre, on utilise la contrainte {sequence} (collection ordonnée permettant les doublons), mais en Java par ex. « sequence » veut dire la même chose que « arraylist ».

Exemples (en Java) d'utilité des associations ordonnées :

- Trier des objets « **Produit** » par prix, en utilisant la méthode **Collections.sort(ArrayList<Produit>)**, mais à condition que la classe **Produit** implémente l'interface **Comparable**. (Le principe d'implémentation sera détaillé dans les prochains paragraphes)



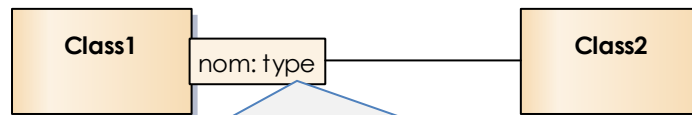
- Trier les lignes d'une **JTable** selon les valeurs de l'une de ses colonnes (Ex. prix, date, prénom, quantité stock, ...), en utilisant la méthode **TableRowSorter.setComparator()** et une implémentation de l'interface **Comparator**.

- **Contrainte de qualification**

Parfois on a besoin de limiter l'étendue d'une association, pour qu'elle implique un nombre réduit d'objets, au lieu d'une multitude d'objets.

Cette limitation doit se faire donc selon une information (souvent une clef de BD ou un identifiant) qui va diriger le choix vers l'objet(s) ciblé(s) par l'association, et exclure les autres objets. Cette information – qui agit comme un filtre – est appelée « **le qualificatif** ». On parle alors d'association « **qualifiée** ».

Une association qualifiée est notée comme ceci :



Nom (et accessoirement type) du qualificatif que doit choisir l'objet **Class1** pour réduire le nombre d'objets **Class2**



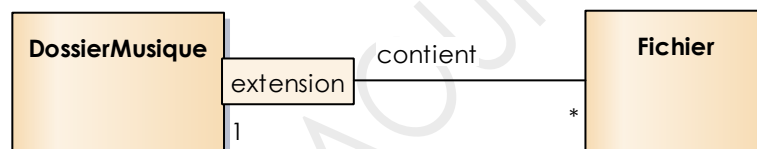
IMPORTANT

- Le qualificatif doit être une propriété appartenant à **Class2**, non à **Class1**.
- La qualification n'est pas obligatoire, elle est utilisée selon les besoins de la classe source.

Exemple 1:

Contrairement aux méthodes de la classe « Dossier », les méthodes de la classe « DossierMusique » ne s'intéressent qu'aux fichiers ayant une extension de type audio (mp3, wma, ...).

La classe « DossierMusique » peut donc filtrer l'association qu'elle a avec la classe « Fichier » selon le qualificatif de l'extension.

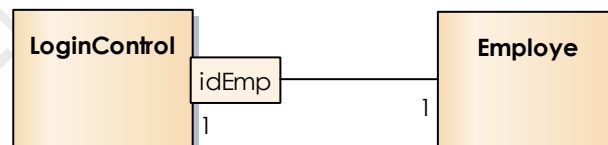


Exemple 2:

Une classe « LoginControl » utilisée comme contrôleur de login, doit être en relation avec la classe « Employé » ayant l'attribut identifiant « idEmp ».

Mais chaque objet « LoginControl » n'a besoin que d'un seul employé, celui dont l'identifiant sera passé en paramètre à ses méthodes.

Donc la classe « LoginControl » peut utiliser le qualificatif « idEmp ».



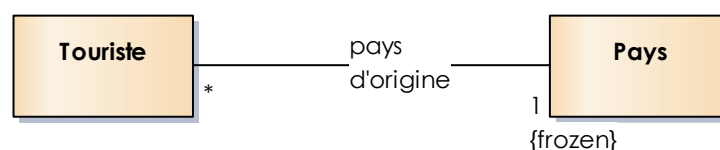
▪ Contrainte de fixation (Associations figées)

Une association entre deux classes **C1** et **C2**, signifie que les objets de ces classes ont un lien mutuel.

Dans certain cas, quand un objet **C1** est lié à un objet **C2**, l'objet **C2** ne peut pas être remplacé par un autre objet **C2**.

On parle alors d'association « **figée** ».

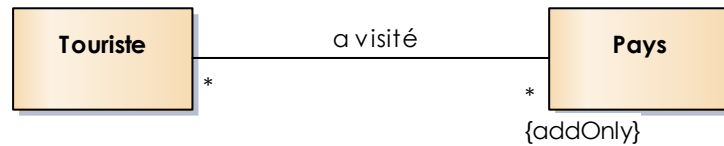
Exemple :



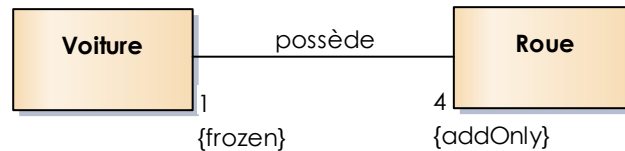
▪ **Contrainte « ajout uniquement » (AddOnly)**

Cette contrainte est imposée quand une classe participe dans une association par un ensemble « **irréductible** » d'objets.

Exemple 1:



Exemple 2:

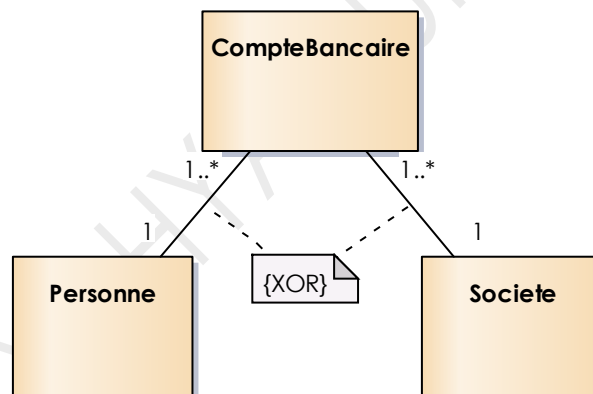


▪ **Contrainte d'exclusion (XOR)**

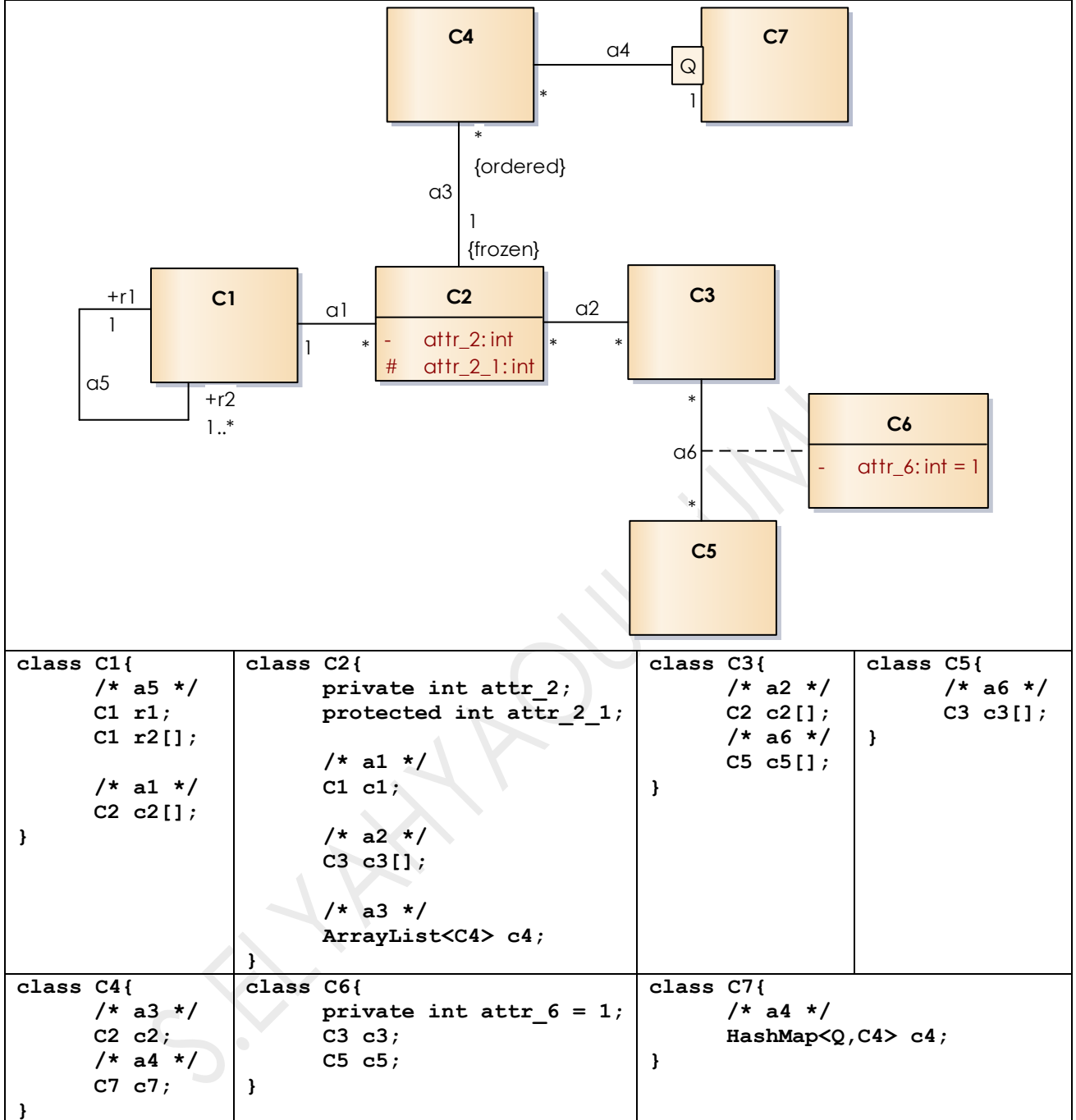
Cette contrainte est imposée quand il y a une **exclusion mutuelle** entre 2 associations (ou plus) qui concernent les mêmes classes.

Exemple :

Un compte bancaire doit appartenir soit à une personne physique, soit à une société.



Synthèse : Correspondances UML ↔ Java



c. L'héritage & les classes abstraites

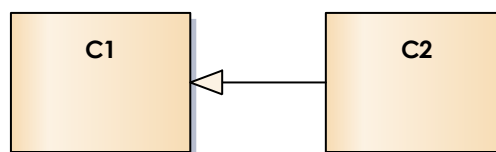
Définition : Héritage

L'héritage est un type spécial d'association qui décrit une sorte de généralisation / spécification entre deux classes, l'une représente le cas général, l'autre représente le cas spécifique.

Quand une classe **C2** hérite d'une autre **C1** – appelée « **classe mère** ou **classe générale** » – cela permet à **C2** – appelée « **classe fille** ou **classe spécifique** » – « d'hériter » de tous les membres **publics** et **protégés sans devoir les déclarer à son tour**.

Evidemment, les membres **privés** ne sont pas hérités car les autres classes ne savent même pas leur existence.

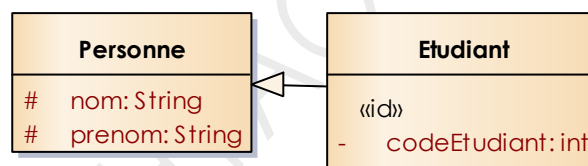
L'héritage est noté par une association à tête vide, sans multiplicités :



Exemple : Héritage Personne / Etudiant

Sémantiquement, les objets de type **Etudiant** sont aussi des objets **Personne**. Mais les objets de type **Etudiant** doivent avoir – **en plus des attributs du type Personne** – un autre attribut **codeEtudiant**.

On sait donc que le cas **spécifique** est **Etudiant**, et le cas **général** est **Personne**.

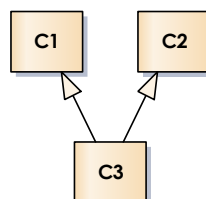


IMPORTANT

Il est essentiel de rappeler que les principes de l'orienté objets ne sont pas implémentés par tous les langages de P.O.O. de la même manière.

Exemple :

Contrairement à d'autres langages, l'héritage multiple est interdit en Java.

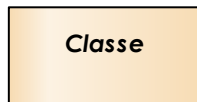


Définition : Classes abstraites

Une classe est dite « **abstraite** » dans les cas suivants :

- Si elle doit être déclarée abstraite par le programmeur pour des raisons sémantiques ou liées à des besoins de programmation.
- Ou si au moins une de ses méthodes est abstraite à son tour. Une méthode abstraite est une méthode qui n'a pas de corps. (Les méthodes seront détaillées dans le chapitre Diagramme de classes de conception).

Les classes abstraites sont notées de deux manières possibles :

Soit en **italique**Soit avec le stéréotype « **abstract** »**IMPORTANT**

En UML, tout élément graphique indiqué en notation italique, est abstrait.

Utilité des classes abstraites

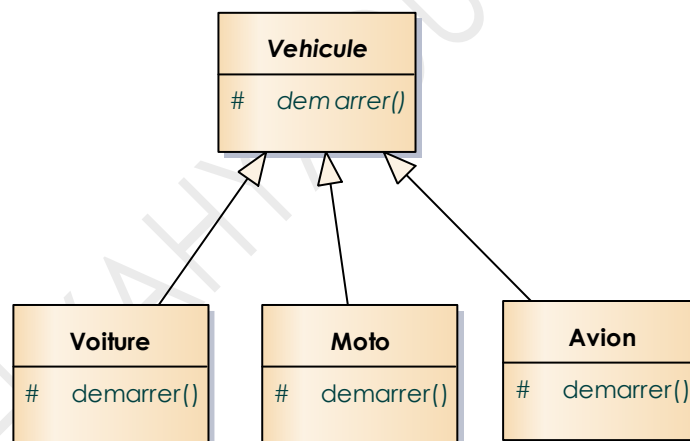
Parfois certains comportements (méthodes) d'une classe ne sont pas concrets.

Le concepteur et le programmeur sont donc obligés de rendre la classe abstraite, car ils ne peuvent pas donner un corps aux méthodes abstraites de la classe. Il incombe aux classes filles de définir ces méthodes.

Exemple :

Le comportement « démarrer » de la classe « Véhicule » n'est pas concret, vu que chaque sous-type de véhicules démarre de manière différente des autres sous-types.

Le concepteur n'a donc pas d'autres choix que de déclarer cette classe abstraite, et d'imposer aux classe filles (Voiture, Moto, ...) de concrétiser la méthode abstraite (démarrer) héritée de leur classe mère.

**Contraintes liées à l'utilisation des classes abstraites**

- Comme son nom l'indique, une classe abstraite ne peut pas être instanciée.
- Une classe qui hérite d'une classe abstraite, doit définir toutes les méthodes abstraites héritées, sinon elle doit être déclarée à son tour abstraite.

d. L'implémentation & les « interfaces »

Définition : Interface

Contrairement à une classe abstraite, une interface **ne peut avoir aucune méthode concrète**.

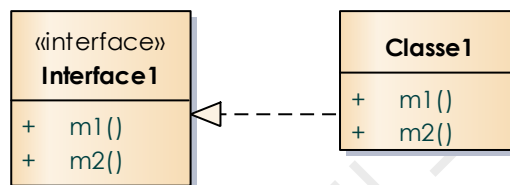
Les interfaces sont représentées avec le stéréotype « interface » comme ceci :



Définition : Implémentation

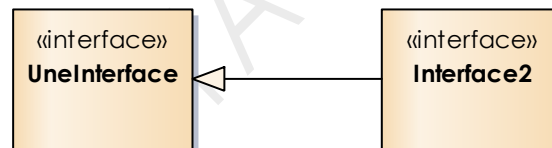
L'association qui lie une classe à une interface est appelée « **implémentation** ».

L'implémentation est notée par une association à tête vide, dont le trait est **discontinu** :



REMARQUE

Une interface peut **hériter** d'une autre interface.



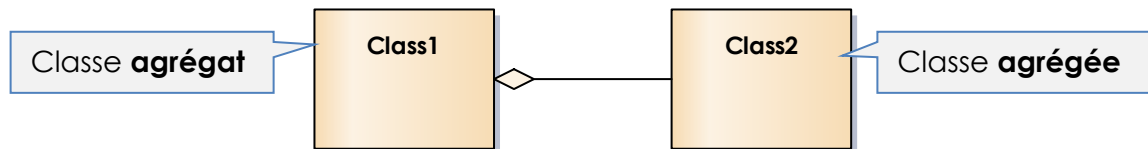
Contraintes liées à l'utilisation des interfaces

- Comme les classes abstraites, les interfaces ne peuvent pas être instanciées.
- Une classe qui implémente une interface, doit définir toutes les méthodes héritées, sinon elle doit être déclarée abstraite.
- Les attributs d'une interface **ne peuvent être ni privés ni protégés**, sont **obligatoirement initialisés à leur déclaration**, et sont **par défaut déclarés constants** (Ex. **final** pour Java).
- Une classe qui implémente une interface, **n'a pas le droit de modifier la valeur d'un attribut hérité**.
- Les méthodes d'une interface **ne peuvent être ni privées, ni protégées ni statiques**.

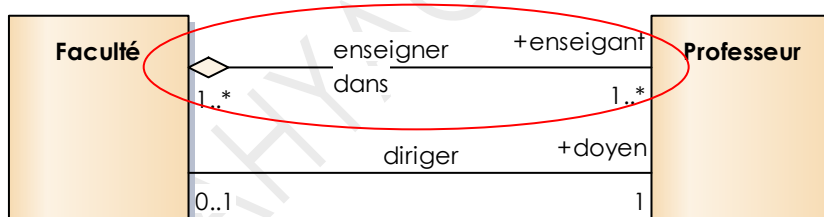
e. Agrégation / Composition**Définition : Agrégation**

L'agrégation est aussi un type d'association en UML. Elle montre une relation « **d'appartenance** », « **d'ensemble / élément** » ou « **agrégat / agrégé** » entre 2 classes. En d'autres termes, l'objet de la classe considérée comme **agrégat**, est constitué d'un ou plusieurs objets de la classe considérée comme **agrégée**.

L'association d'agrégation est représentée par un losange vide :

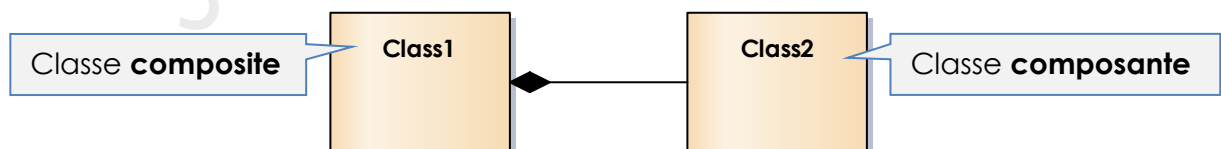
**IMPORTANT**

L'agrégation entre 2 classes **n'implique aucun caractère d'exclusivité**, c.à.d. que l'objet agrégé **peut être partagé par plusieurs agrégats**.

Exemple 1**Exemple 2****Définition : Composition**

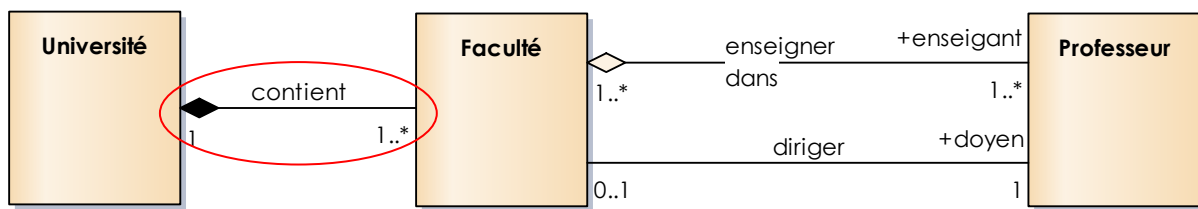
La composition est un cas spécial de l'agrégation. Elle est considérée comme une « **agrégation forte** ».

L'association de composition est représentée par un losange plein :

**IMPORTANT**

L'agrégation entre 2 classes **implique directement un caractère d'exclusivité**, c.à.d.

- L'objet composant **appartient à un seul objet composite**, ce qui signifie que **la multiplicité de la classe composite est obligatoirement 1**.
- **Si l'objet composite est détruit, alors l'objet composant aussi doit être détruit.**

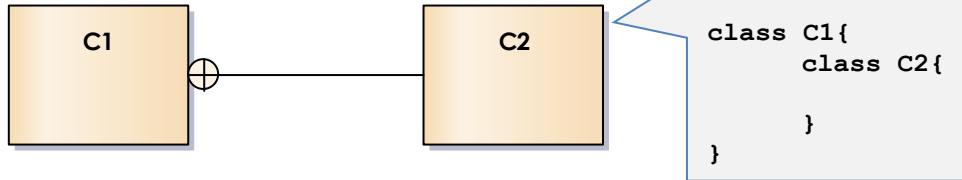
Exemple**Agrégation / Composition ↔ POO**

- Lors du passage du modèle UML vers le modèle POO, agrégation et composition sont traitées de la même manière qu'une simple association.
- **Uniquement pour la composition :**
La destruction du composite doit être accompagnée de la destruction des composants.

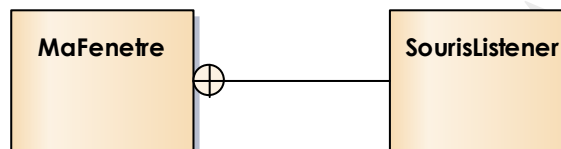
f. Les classes internes (Inner or Nested classes)**Définition**

En P.O.O. une classe **C2** est dite « **interne** » à une autre classe **C1**, quand **C2** est définie à l'intérieur de la définition de **C1**.

Cette association est sans multiplicité.

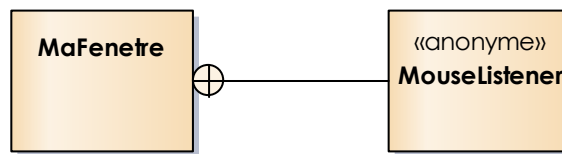


L'une des utilisations les plus répandues de cette association est la notion de « classe interne anonyme », souvent utilisée pour la création « d'écouteurs d'événements » (Event Listeners) en programmation événementielle d'interface graphiques Windows.

Exemple 1 : Ecouteur d'événement de la souris avec classe interne

```

Public class MaFenetre extends JFrame{
    // constructeur
    public MaFenetre() {
        this.addMouseListener(new SourisListener());
    }
    // classe interne nommée « SourisListener »
    Public class SourisListener implements MouseListener {
        public void mouseClicked(MouseEvent e) {
            if(e.getClickCount() == 2)
                System.out.println("double-clique !");
            if(e.getClickCount() == 1)
                System.out.println("simple clique !");
        }
        public void mousePressed(java.awt.event.MouseEvent e) {
        }
        public void mouseReleased(java.awt.event.MouseEvent e) {
        }
        public void mouseEntered(java.awt.event.MouseEvent e) {
        }
        public void mouseExited(java.awt.event.MouseEvent e) {
        }
    }
}
  
```

Exemple 2 : Ecouteur d'événement de la souris avec classe interne anonyme

```

class MaFenetre extends JFrame{
    // constructeur
    public MaFenetre(){
        this.addMouseListener(/*classe interne anonyme*/
            new MouseListener() {
                public void mouseClicked(java.awt.event.MouseEvent e) {
                    if(e.getClickCount() == 2)
                        System.out.println("double-clique !");
                    if(e.getClickCount() == 1)
                        System.out.println("simple clique !");
                }
                public void mousePressed(java.awt.event.MouseEvent e) {
                }
                public void mouseReleased(java.awt.event.MouseEvent e){
                }
                public void mouseEntered(java.awt.event.MouseEvent e) {
                }
                public void mouseExited(java.awt.event.MouseEvent e) {
                }
            }
        );
    }
}

```

Etude de cas :**Diagramme de classes : Système de fichiers**

[.....

Un dossier ou répertoire peut contenir plusieurs autres dossiers et plusieurs fichiers, mais il peut aussi être vide.

Un dossier ne peut avoir qu'un seul dossier parent.

Un fichier aussi ne peut avoir qu'un seul dossier parent.

Un raccourci est un type spécial de fichiers.

Un raccourci peut designer soit un seul fichier soit un seul dossier.

Fichiers et dossiers peuvent avoir plusieurs raccourcis.

La suppression d'un dossier, implique la suppression de tous ses dossiers et fichiers, mais pas celle des raccourcis qui pointent vers ce dossier.

Les dossiers et fichiers contenus dans un dossier doivent être ordonnés.

.....]

Travail à faire :

- Réaliser le diagramme de classes

3. Réaliser un diagramme de classes à partir d'un cahier des charges

Etapes à suivre :

- 1) Dictionnaire de données
- 2) Dépendances fonctionnelles
- 3) Classes
- 4) Associations
- 5) Normalisation

Ces étapes seront étudiées un à une à travers l'étude de cas : « Gestion d'agence immobilière ».

Etude de cas : Gestion d'agence immobilière **Cahiers des charges**

[...

L'agence immobilière doit connaître l'implantation de chaque logement (identifiant de la commune, nom de la commune, identifiant et nom du quartier), et pour chaque commune, on désire connaître le nombre d'habitants ainsi que la distance séparant la commune de l'agence.

Chaque logement doit avoir un identifiant, une adresse complète (exemple : rue ou boulevard + immeuble + étage + numéro), une superficie ainsi que le montant du loyer.

Chaque logement appartient à un type de logement bien défini (maison, studio, appartement, villa, ...)

L'agence facturera en plus du loyer, une somme forfaitaire à ses clients, appelée : « charge forfaitaire ».

Ces charges forfaitaires dépendent du type de logement. Par exemple, le prix d'un studio sera égal au prix du loyer + 50 DH (valeur minimale) de charges forfaitaires par mois, tandis que pour un appartement en immeuble la valeur sera loyer + 100 DH.

Quant au signataire du contrat de bail, le système doit contenir : CIN, nom, prénom, adresse, date de naissance et numéro de téléphone.

Pour chaque contrat, on désire avoir le numéro de location, ainsi que la date de location. Un contrat peut concerner plusieurs logements s'il s'agit du même locataire.

La valeur du loyer d'un logement peut changer d'un contrat à un autre.

...]

a. Etablir le dictionnaire de données

Définition				
<p>Le dictionnaire de données est un inventaire qui regroupe les spécifications relatives (<i>métadonnées</i>) à chaque donnée : nom de la donnée, signification, type (chiffre, chaîne, ...), valeur par défaut, ...</p> <p>Les données regroupées doivent respecter les principes suivants :</p> <ul style="list-style-type: none"> ▪ Eliminer les synonymes Ex. : <i>type logement</i> ↔ <i>catégorie logement</i> ▪ Eliminer les données « calculées » : Qui peuvent être obtenues à partir d'autres données. Ex. : <i>âge</i> ↔ <i>date de naissance</i> ▪ Eliminer les regroupements de données Ex. : <i>[adresse]</i> ↔ <i>[n° + rue + ville + code postal + pays]</i> 				
Etude de cas : Agence immobilière				
Dictionnaire des données				
#	Donnée	Signification	Type	Commentaires
1	idCommune	Identifiant de la commune	Nombre entier	Numéro automatique qui s'incrémente après chaque ajout
2	nomCommune	Nom de la commune où se trouve le logement	Texte	Ex. : "Agdal"
3	distAg	Distance qui sépare la commune de l'agence	Nombre réel	Ex. : 3,5KM
4	nbrHab	Nombre d'habitants de la commune	Nombre entier	
5	idQuart	Identifiant du quartier où se trouve le logement	Nombre entier	Numéro automatique
6	quartier	Nom du quartier	Texte	
7	idLogement	Identifiant du logement	Nombre entier	Numéro automatique
8	Loyer	Montant du loyer du logement	Nombre réel	
9	superficie	Superficie du logement	Nombre réel	
10	adresseLog	Adresse complète du logement	Texte	
11	nom	Nom du locataire	Texte	
12	prénom	Prénom du locataire	Texte	
13	adresse	Adresse du locataire	Texte	
14	tel	N° de téléphone du locataire	Texte	Si le numéro de téléphone est considéré comme un entier, le 1 ^{er} zéro ne sera pas pris en compte par le système
15	CIN	Identifiant du locataire	Texte	
16	dateNaiss	Date de naissance du locataire	Texte	Ex. : "11/05/1956"
17	N°Location	Numéro de location	Nombre entier	

18	dateLoc	Date du contrat de location	Texte	Ex. : "01/01/2010 – 10:30" L'heure est demandée pour le cas des locations par journée
19	nomType	Type du logement	Texte	Ex. villa, bureau, appartement, ...
20	chargesForf	Charges forfaitaires	Nombre réel	Valeur par défaut : 50,00

b. Définir les dépendances fonctionnelles**Définition : Dépendance fonctionnelle**

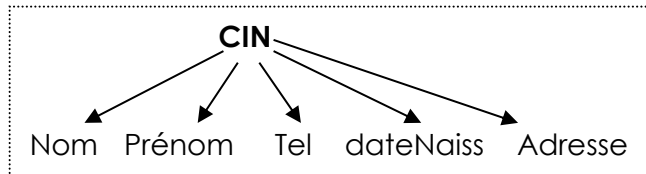
On dit qu'une donnée B est en dépendance fonctionnelle (DF) avec une autre donnée A, si la valeur de A défini de manière unique la valeur de B.

La DF est notée comme ceci : $A \longrightarrow B$

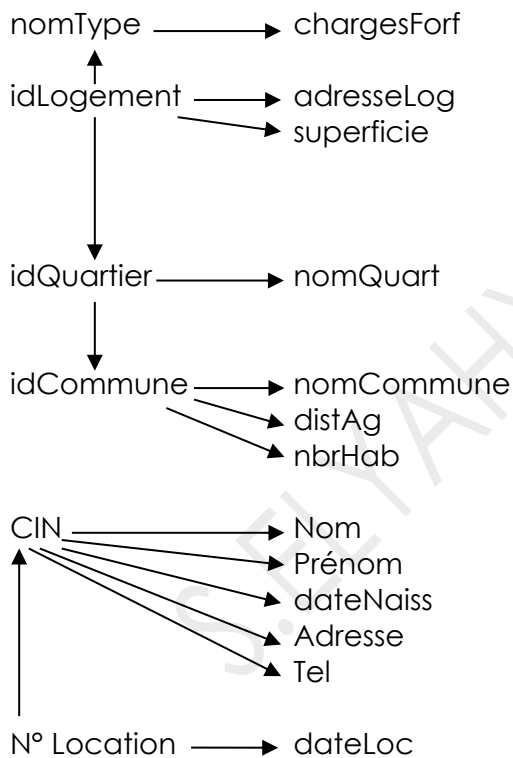
Etude de cas Agence immobilière

Pour chaque locataire, la valeur de la donnée CIN définit de manière unique la valeur de la donnée Nom. On dit donc qu'il y a une dépendance fonctionnelle entre les données CIN et Nom.

On écrit $CIN \longrightarrow Nom$



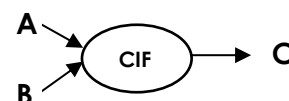
Liste des dépendances fonctionnelles :

**Définition : Contrainte d'intégrité fonctionnelle (CIF)**

Une CIF est une DF dont l'origine est un ensemble de données au lieu d'une donnée unique.

Si la valeur d'une donnée C est définie de manière unique celle du couple (A, B), on dit que cette DF est une CIF

La CIF est notée comme ceci $(A,B) \longrightarrow C$ ou bien :

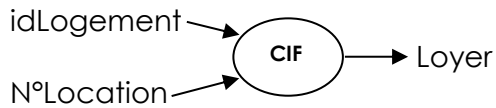


Étude de cas Agence immobilière : (idLogement, N°Location) → Loyer

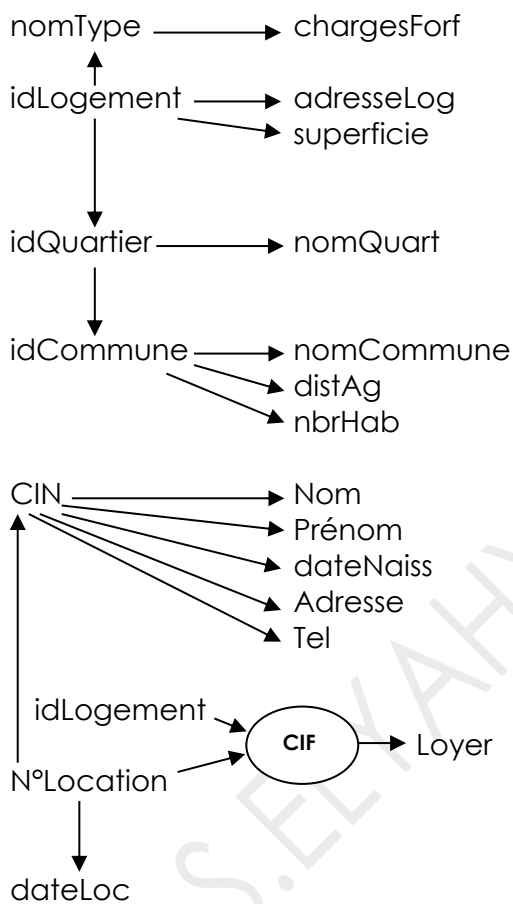
La donnée **Loyer** ne peut exister que par une liaison entre un contrat (**N°Location**) et un logement (**idLogement**).

On peut donc dire que *loyer* dépend à la fois de *N°Location* et *idLogement*

Alors :

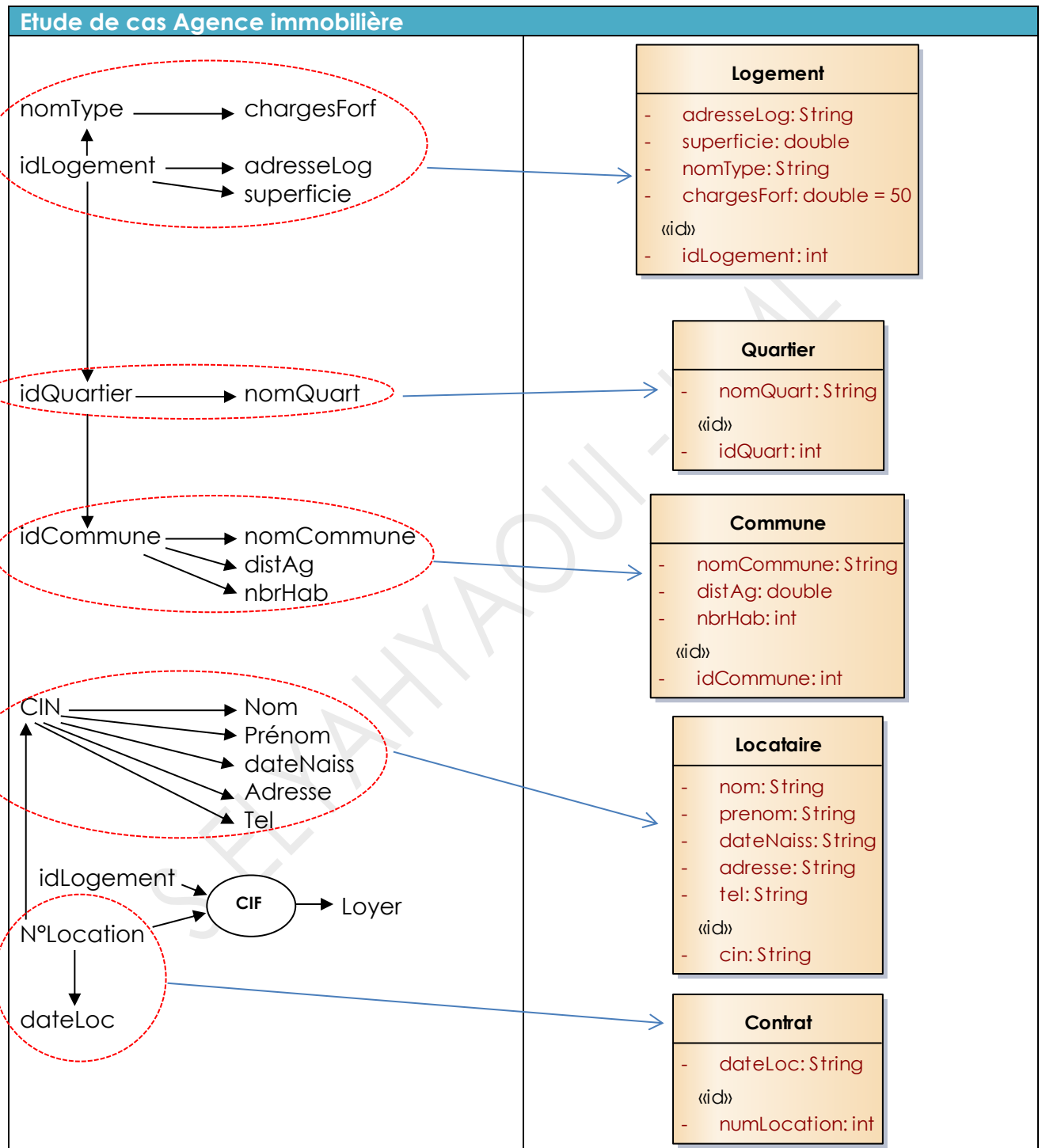


Liste des dépendances fonctionnelles mise à jour:



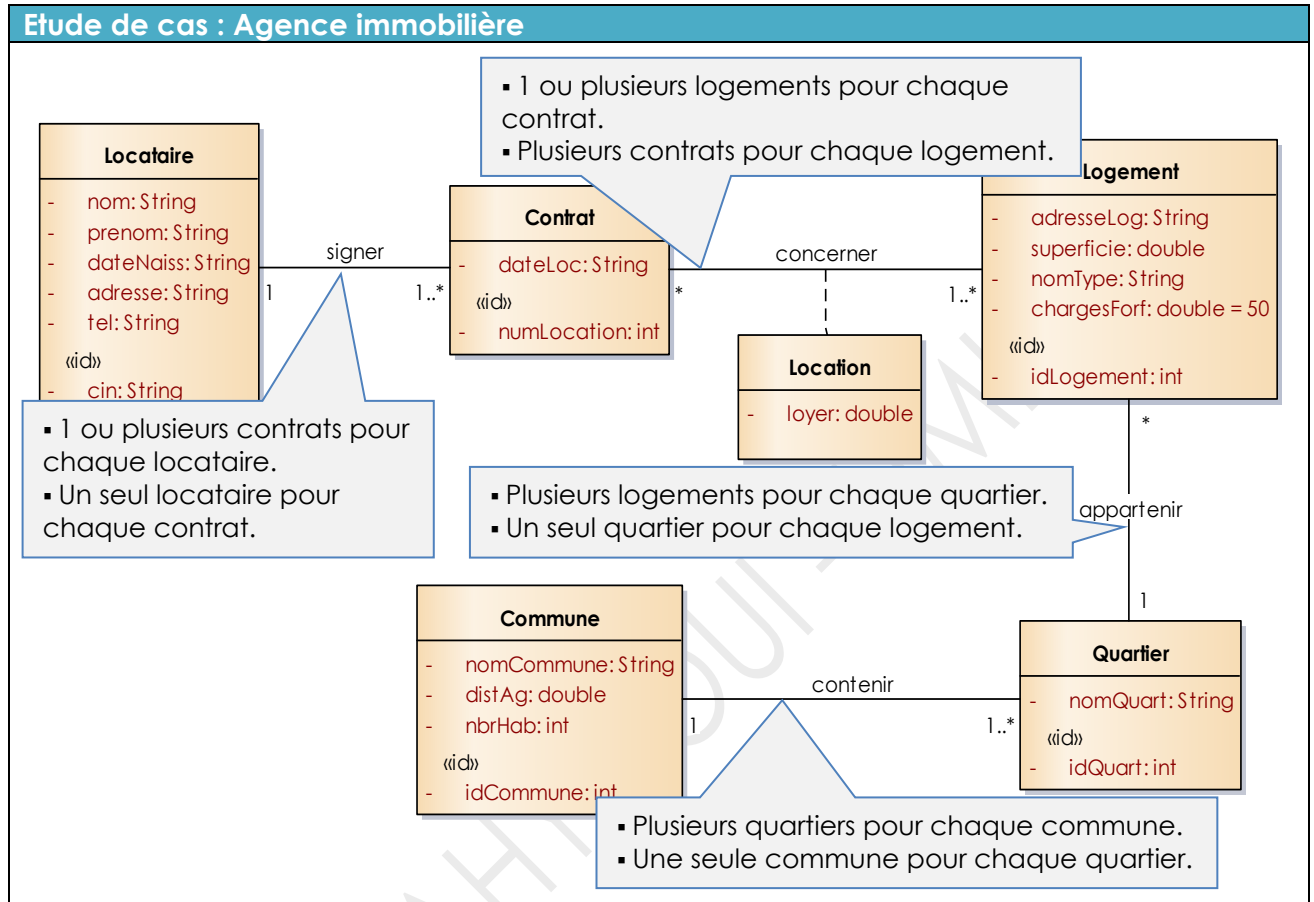
c. Construire les classes

Chaque regroupement de données appartenant à la même famille constituera une classe.



d. Définir les associations et multiplicités

- Les dépendances entre identifiants donneront lieu à des associations.
- Les données dépendantes des CIF seront logées dans des classes d'association.



e. Normalisation du DCL

→ Forme normale n°1

Chaque classe doit obligatoirement avoir un identifiant. Cet identifiant peut être composé de plusieurs attributs (rarement), dans ce cas ces attributs sont précédés par le stéréotype « id ».

Exemple :

Classe
Attribut_1 Attribut_2 « id » Attribut_3 Attribut_4

→ Forme normale n°2

Pour les classes ayant un identifiant formé de plusieurs attributs, les autres attributs doivent dépendre de l'identifiant tout entier, et non d'une partie de cet identifiant.

Exemple :

Classe
Attribut_1 Attribut_2 « id » Attribut_3 Attribut_4

Si Attribut_3 → Attribut_2 au lieu de (Attribut_3, Attribut_4) → Attribut_2, alors cette classe ne respecte pas la FN n°2.

→ Forme normale n°3

Les attributs doivent dépendre de l'identifiant de manière directe, sans aucune transitivité.

Exemple :

Classe
Attribut_1 Attribut_2 « id » Attribut_0

Si Attribut_0 → Attribut_1 et Attribut_1 → Attribut_2, alors Attribut_2 ne dépend pas directement de l'identifiant, cette classe ne respecte donc pas la FN n°3.

→ Forme normale de Boyce-Codd

Pour les identifiants composés de plusieurs attributs, ces derniers ne doivent pas être dépendants d'un autre attribut.

Exemple :

Classe
Attribut_1
Attribut_2
« id »
Attribut_3
Attribut_4

Si Attribut_2 → Attribut_3 alors cette classe ne respecte pas la FN BC.

Étude de cas : Agence immobilière

Reprenons la classe **Logement**.

On remarque la transitivité suivante :

idLogement → **nomType** → **chargesForf**

Logement
- adresseLog: String
- superficie: double
- nomType: String
- chargesForf: double = 50
«id»
- idLogement: int

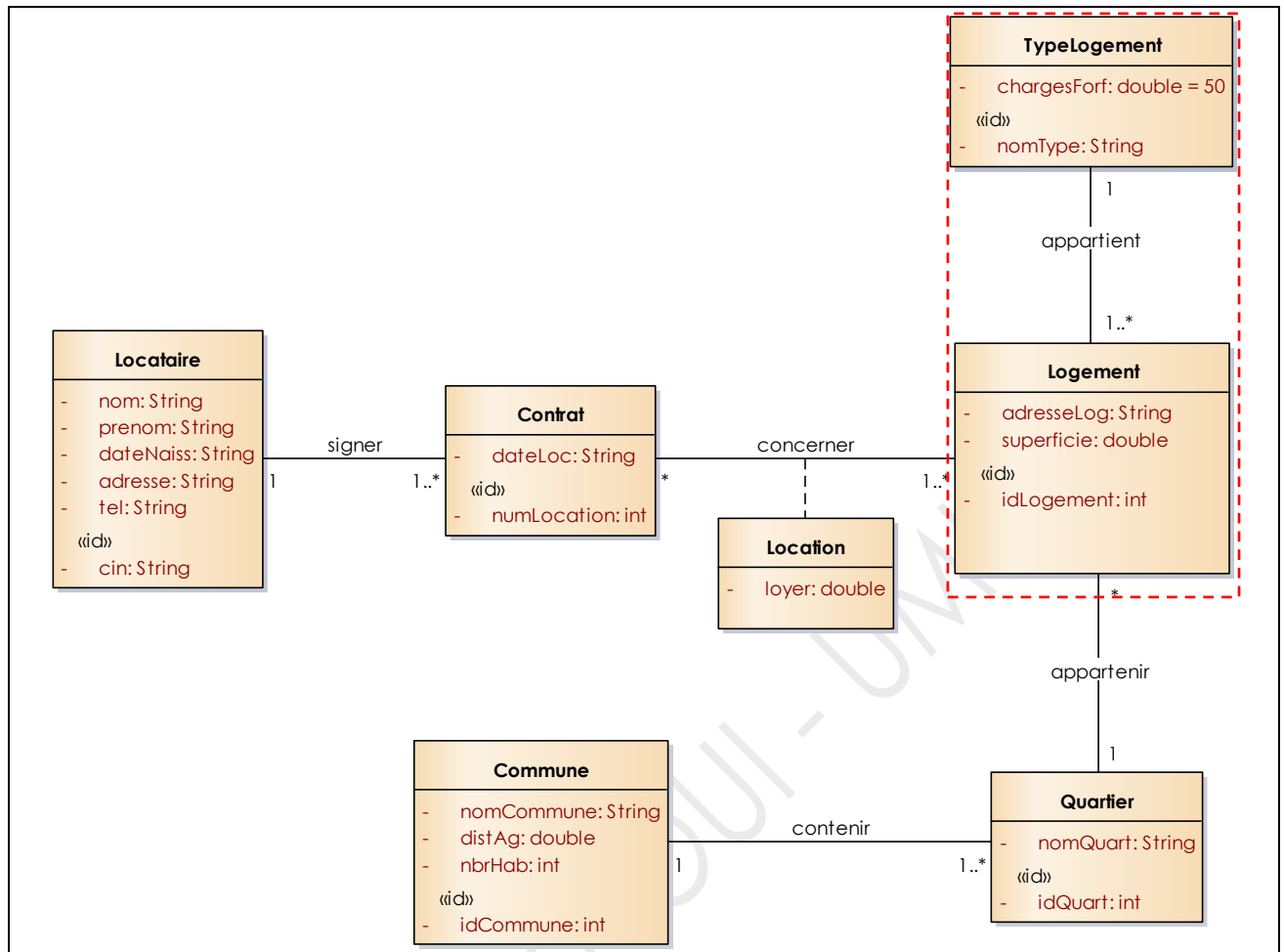
La **FN3** n'est donc pas respectée.

→ Solution proposée :

Généralement, dans le cas de non-respect de la **FN3**, la solution indiquée est de séparer la classe en deux classes qui seront liées par une association.

On nommera la nouvelle classe : **TypeLogement**.

Bien sûr, il est inapproprié de migrer les 3 attributs vers la classe **TypeLogement**, car dans ce cas la classe **Logement** perdra son identifiant. La solution est donc de migrer les attributs **nomType** et **chargesForf**.



Etudes de cas :

Gestion commerciale – suite

Une société de vente souhaite créer un système d'informations pour gérer ses transactions.

Pour chaque commande, le système doit contenir la date (AAAA-MM-JJ) et le numéro de la commande, les informations du client (*nom, prénom, adresse et numéro de téléphone*), des produits commandés (*numéro, référence, désignation, prix unitaire et quantité stock*), et la quantité commandée de chaque produit.

Une commande contient plusieurs produits, si l'un de ces produits n'existe pas en quantité suffisante en stock, l'un des responsables de ventes doit alors enregistrer une demande d'alimentation du stock avec la quantité nécessaire. Pour chacune de ces demandes, on souhaite connaître le responsable de ventes, la commande client concernée, les produits et leurs quantités demandées, la date d'enregistrement de la demande et le fournisseur.

Pour chaque livraison, le système doit enregistrer les informations de la livraison effectuée : date & heure (AAAA-MM-JJ hh-mm), numéro de la commande, informations du livreur (*numéro, nom, prénom, adresse*).

Chaque commande réglée par le client doit avoir une (unique) facture. On doit connaître le numéro de la facture et le type de paiement (espèces, cheque, virement bancaire) et la date de facturation.

S'il s'agit du même client, une facture peut concerner une ou plusieurs commandes.

Travail à faire :

- DCL.

Système de réservation de vols pour agence de voyages

Une agence de voyages désire gérer les informations concernant les réservations de vols pour ces clients dans une application à part.

L'agence traite avec plusieurs compagnies aériennes. Chacune de ces dernières doit avoir un nom et un siège social.

Une compagnie aérienne propose plusieurs vols chaque jour.

En général, un vol n'appartient qu'à une seule compagnie. Mais s'il s'agit d'un vol avec escales, il peut appartenir à plusieurs compagnies.

Chaque vol est caractérisé par un numéro de vol, une date de départ (JJ-MM-AA hh:mm), une date d'arrivée, une durée officielle (qui doit être préalablement connue par les clients), une durée effective (en tenant compte des éventuels retards) et un nombre maximal de passagers.

Pour les vols avec escales, on doit aussi connaître le nombre d'escales.

Un vol doit également avoir :

- Un aéroport de départ (en précisant le nom du terminal de départ),
- Un aéroport d'arrivée (en précisant le nom du terminal d'arrivée),
- Et – pour les vols avec escales – un ou plusieurs aéroports d'escale (en précisant le nom du terminal de l'escale).

Pour chaque escale, il faut préciser la date de départ, la date d'arrivée et le type :

escale prévue / escale forcée (à cause de mauvaises conditions météo par ex.).

Pour chaque aéroport on doit connaître le nom, et le nombre de terminaux.

Bien sûr, pour prendre un vol, il faut une réservation.

Une réservation concerne un seul vol, et il peut arriver qu'un vol ne reçoive aucune réservation.

Chaque réservation doit avoir un numéro de réservation et une date.

Un client, avant de pouvoir passer une réservation (ou plusieurs), doit d'abord renseigner le système sur ses informations (email, nom, prénom, adresse, téléphone et identifiant bancaire).

Une réservation appartient à un et un seul client. Par contre elle concerne au moins un passager. Le client doit également renseigner toutes les informations des passagers (nom, prénom, sexe, âge).

L'application doit aussi traiter les réservations de vols avec réservation de chambre d'hôtel.

Ce genre de réservations contient des informations supplémentaires (par rapport à une réservation ordinaire). On doit connaître le nom de l'hôtel, le type de chambre (simple, double, ...) et la date de début et de fin du séjour.

Travail à faire :

- **DCL.**

Gestion de réservations d'hôtel – suite

[.....

La direction générale d'une chaîne d'hôtels souhaite créer un système informatique généralisé, pour la gestion interne de tous les hôtels de la chaîne, les chambres, les clients et les réservations.

Pour chaque hôtel on doit connaître le code, l'adresse, la ville, le nombre de chambres et de suites, et le chiffre d'affaires mensuel et annuel.

Un hôtel possède au moins 10 chambres et 2 suites (chambres dont la superficie dépasse 70m²).

Chaque chambre est identifiée par un code, mais on doit aussi connaître son état (libre/réservée/habitée/en travaux), le numéro, le nombre de lits simples, le nombre de lits doubles, la superficie, le nombre de balcons et salles de bain, et le prix officiel de la nuitée (même si le prix réel varie selon les saisons et les promotions).

Pour réserver une chambre, le client doit fournir le jour d'arrivée, le jour de départ, le nombre de personnes adultes et enfants qui vont séjourner dans la chambre, et aussi ses coordonnées à lui : nom et numéro de téléphone.

Le système doit aussi enregistrer la date de réservation.

Bien sûr, une réservation concerne une seule chambre.

Les informations des personnes hébergées, seront renseignées à leur arrivée. Ces informations sont : CIN (si adulte), nom et prénom, âge, profession et sexe.

Chaque chambre contient (dans le réfrigérateur) des produits (jus & boissons alcooliques

par ex.). Pour chaque produit on doit avoir les informations suivantes : l'identifiant, la désignation et le prix unitaire.

Les prix des produits consommables présents dans les chambres, ne sont pas inclus dans le prix du séjour. Ce qui veut dire que chaque séjour peut avoir une facture supplémentaire qui doit non seulement montrer le prix total de la consommation, mais aussi la quantité consommée de chaque produit.
Une facture ne concerne qu'un seul séjour.

La gestion des employés de la chaîne doit aussi figurer dans le système (directeurs, gérants, réceptionnistes, femmes de ménage, ...).

On doit connaître les informations de chaque employé : CIN, date de naissance, salaire, type de poste, date d'embauche, adresse, téléphone, et bien sûr nom et prénom.
Un hôtel n'a qu'un seul directeur, et plusieurs employés.

Même si un employé peut être muté d'un hôtel à un autre, il ne peut travailler que dans un seul hôtel.

Donc, en cas de mutation d'un employé, on doit connaître la date et le motif de mutation.

.....]

Travail à faire :

- DCL.