



# DIAPORAMAS

(SUPPORT ELECTRONIQUE DE COURS)

## GENIE LOGICIEL



PROFESSEUR : LAHCEN MOUMOUN

DEPARTEMENT GENIE INFORMATIQUE &  
MATHEMATIQUES





## Module : Génie logiciel

### Chapitre 1: Introduction Génie logiciel

Pr LAHCEN MOUMOUN  
[ensablearn@gmail.com](mailto:ensablearn@gmail.com)

Département Génie Informatique & Mathématiques

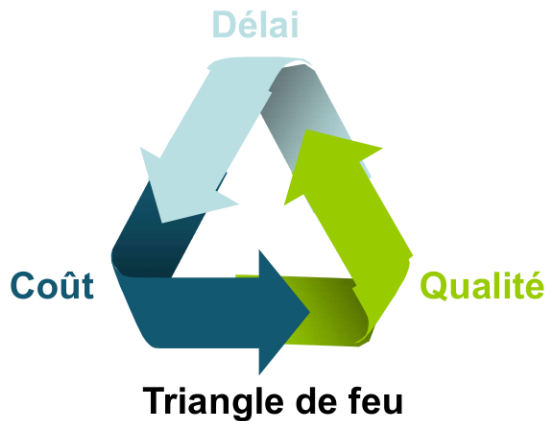
## Le génie logiciel

- **Le génie logiciel** est l'application d'une approche systématique, disciplinée et quantifiable au développement, à l'exploitation et à la maintenance des logiciels (Définition IEEE)
- **Le génie logiciel** est l'ensemble des **activités** de conception et de mise en œuvre et des **méthode** et **procédures** tendant à rationaliser la production du logiciel et son suivi.
- Il s'agit d'une fabrication **collective** concrétisée par un ensemble de documents de conception, de programmes et de jeux de tests avec souvent de **multiples versions**.

## Le génie logiciel

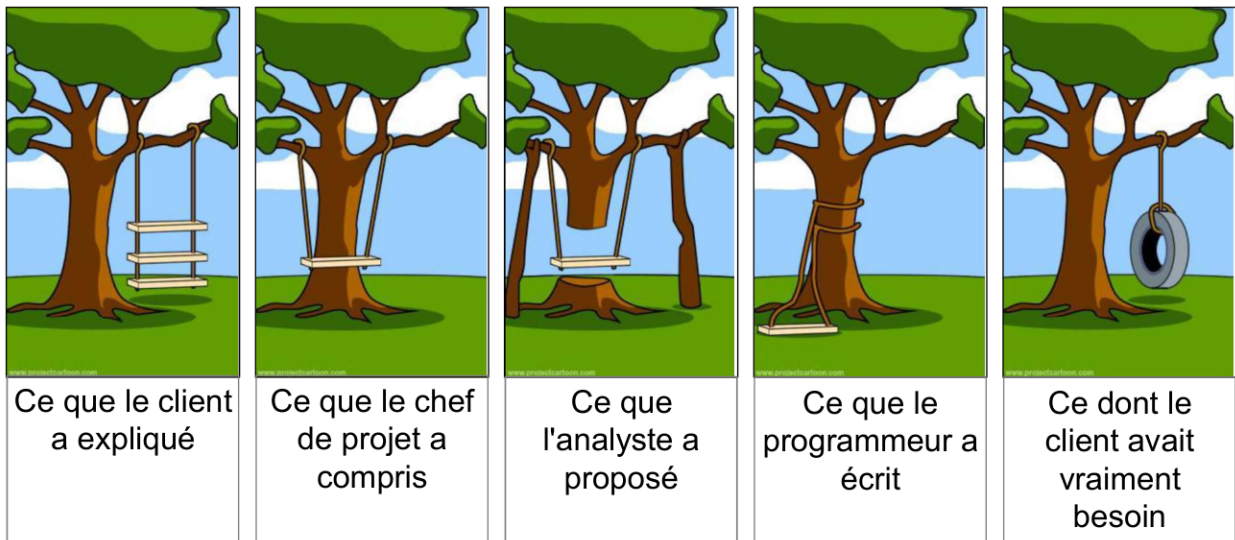
Le **génie logiciel** s'intéresse en particulier aux procédures qui permettent de produire des logiciels qui

- ◆ Correspondent aux attentes du client, aux **besoins**
- ◆ Soient **fiables et** aient de **bonnes performances**
- ◆ Respectent les **délais** et les **coûts de construction** et un **coût d'entretien réduit**



- **Validité** : réponse aux besoins des utilisateurs
- **Facilité d'utilisation** : prise en main et robustesse
- **Performance** : temps de réponse, débit, fluidité...
- **Fiabilité** : tolérance aux pannes
- **Sécurité** : intégrité des données et protection des accès
- **Maintenabilité** : facilité à corriger ou transformer le logiciel
- **Portabilité** : changement d'environnement matériel ou logiciel

## Le génie logiciel



Les clients ne savent pas ce dont ils ont besoin. Le prototypage (maquette pour l'interface utilisateur) est souvent utilisé pour illustrer ce besoin.

- Le développement de logiciels comprend toujours les activités suivantes (dans une certaine mesure) :
  - Analyse des besoins
  - Concevoir
  - Construction
  - Tester (parfois)
- Ces activités ne s'enchaînent pas strictement les unes après les autres !
- Se chevauchent et interagissent souvent

- Les exigences logicielles définissent la fonctionnalité du système
  - Répondre à la question « quoi ? » et non « comment ? »
  - Définir les contraintes du système
- Deux types d'exigences
  - Exigences fonctionnelles
  - Exigences non fonctionnelles

# L'analyse des besoins

- L'analyse des besoins part d'une vision du système
  - Les clients ne savent pas ce dont ils ont besoin !
  - Les exigences viennent grossièrement et sont spécifiées et étendues de manière itérative
- Le prototypage est souvent utilisé, notamment pour l'interface utilisateur
- Le résultat est la spécification des exigences logicielles (SRS)

# Spécification des exigences logicielles

- La spécification des exigences logicielles (SRS) est un document formel d'exigences
- Il décrit en détails :
  - Exigences fonctionnelles
    - Processus d'affaires
    - Acteurs et cas d'utilisation
  - Exigences non fonctionnelles
    - Par exemple. performances, évolutivité, etc.

- Il est toujours difficile de décrire et de documenter les exigences de manière exhaustive et non ambiguë
  - De bonnes exigences permettent d'économiser du temps et de l'argent
- Les exigences changent toujours au cours du projet !
  - Une bonne spécification des exigences logicielles réduit les changements
  - Les prototypes réduisent considérablement les changements

- La **conception du logiciel** est une description technique de la manière dont le système mettra en œuvre les exigences
- **L'architecture du système** décrit :
  - Comment le système sera décomposé en sous-systèmes (modules)
  - Responsabilités de chaque module
  - Interaction entre les modules
  - Plateformes et technologies

- Le document de conception logicielle (SDD) est une description formelle de l'architecture et de la conception du système
- Il contient:
  - Conception architecturale
    - Les modules et leur interaction (schéma)
  - Pour chaque module
    - Conception de processus (schémas)
    - Conception des données (diagramme E/R)
    - Conception d'interfaces (diagramme de classes)

- Pendant la phase de construction du logiciel, les développeurs créent le logiciel
  - Parfois appelée phase de mise en œuvre
- Il comprend:
  - Conception de méthode interne
  - Écrire du code
  - Rédaction de tests unitaires (parfois)
  - Test et débogage
  - L'intégration



- Le codage est le processus d'écriture du code de programmation (le code source)
  - Le code suit strictement la conception
  - Les développeurs effectuent la conception de méthodes internes dans le cadre du codage
- Le code source est le résultat du processus de construction du logiciel
  - Écrit par des développeurs
  - Peut inclure des tests unitaires

- Les tests vérifient si le logiciel développé est conforme aux exigences
  - Vise à identifier les défauts (bugs)
- Les développeurs testent le code après l'avoir écrit
  - Lancez-le au moins pour voir les résultats
  - Les tests unitaires, c'est encore mieux
    - Les tests unitaires peuvent être répétés plusieurs fois
- Les tests du système sont effectués par des ingénieurs QA
  - Les tests unitaires sont effectués par les développeurs

- Le débogage vise à trouver la source d'un défaut déjà identifié et à le corriger
  - Réalisé par les développeurs
- Étapes de débogage :
  - Trouver le défaut dans le code
    - Identifier la source du problème
    - Identifier l'endroit exact dans le code qui le provoque
  - Corriger le défaut
  - Testez pour vérifier si le correctif est correct

- L'intégration consiste à assembler toutes les pièces
  - Compiler, exécuter et déployer les modules en tant que système unique
  - Test pour identifier les défauts
- Stratégies d'intégration
  - Big bang, du haut vers le bas et du bas vers le haut
  - Intégration continue

- Les développeurs inexpérimentés considèrent le codage comme le cœur du développement
  - Dans la plupart des projets, le codage ne représente que 20 % des activités du projet !
  - Les décisions importantes sont prises lors de l'analyse des besoins et de la conception
  - La documentation, les tests, l'intégration, la maintenance, etc. sont souvent décriés
- L'ingénierie logicielle n'est pas seulement du codage !
  - Programmeur != ingénieur logiciel

- Qu'est-ce que la vérification logicielle ?
  - Il vérifie si le logiciel développé est conforme aux exigences
  - Effectué par les ingénieurs d'assurance qualité du logiciel (QA)
- Deux approches :
  - Examens et inspections formels
  - Différents types de tests
- Impossible de certifier l'absence de défauts !
  - Ne peuvent que diminuer leurs tarifs

## Test du logiciel

- Les tests vérifient si le logiciel développé est conforme aux exigences
- Les tests visent à trouver des défauts (bugs)
  - Tests boîte noire et boîte blanche
  - Tests unitaires, tests d'intégration, tests système, tests d'acceptation
  - Tests de résistance, tests de charge, tests de régression
  - Les ingénieurs testeurs peuvent utiliser des outils de test automatisés pour enregistrer et exécuter des tests

## Processus de test logiciel

- Planification des tests
  - Établir la stratégie de test et le plan de test
  - Pendant les phases d'exigences et de conception
- Développement de tests
  - Procédures de test, scénarios de test, cas de test, scripts de test
- Exécution des tests
- Rapport d'essai
- Retester les défauts

## Module : Génie logiciel

### Chapitre 2: Démarche UP-XP et UML

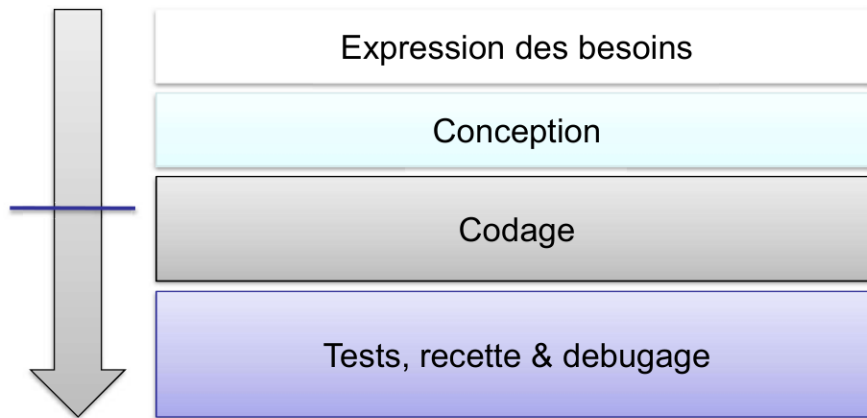
Pr LAHCEN MOUMOUN  
[ensablearn@gmail.com](mailto:ensablearn@gmail.com)

Département Génie Informatique & Mathématiques

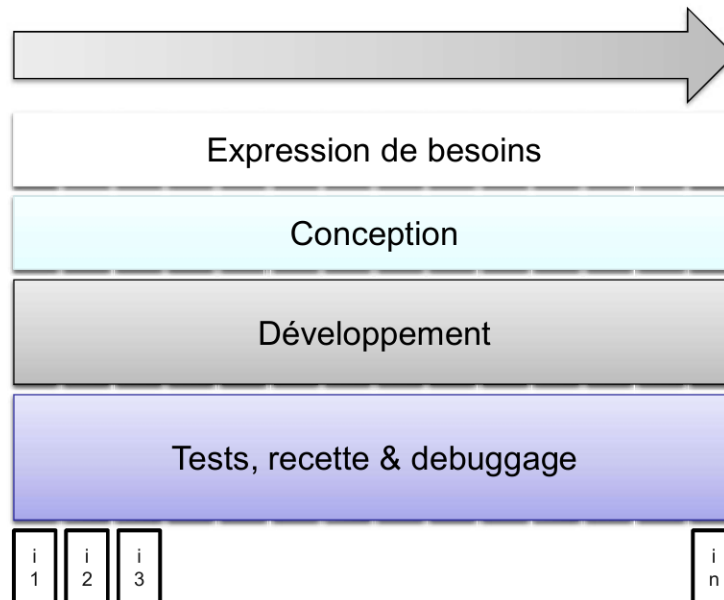
#### Approche agile

- ◆ Approche réactive et itérative d'organisation de travail
- ◆ Focalisée sur la fonctionnalité et satisfaction client
- ◆ Construit en adéquation avec les capacités et limites humaines
  
- ◆ 'Agile' regroupe plusieurs méthodologies :
  - Unified Process (UP)
  - Extreme Programming (XP)
  - Scrum
  - DSDM
  - Crystal
  - ...

## Cycle de vie traditionnel



## Les solutions Agiles



Toujours focalisées sur le produit final

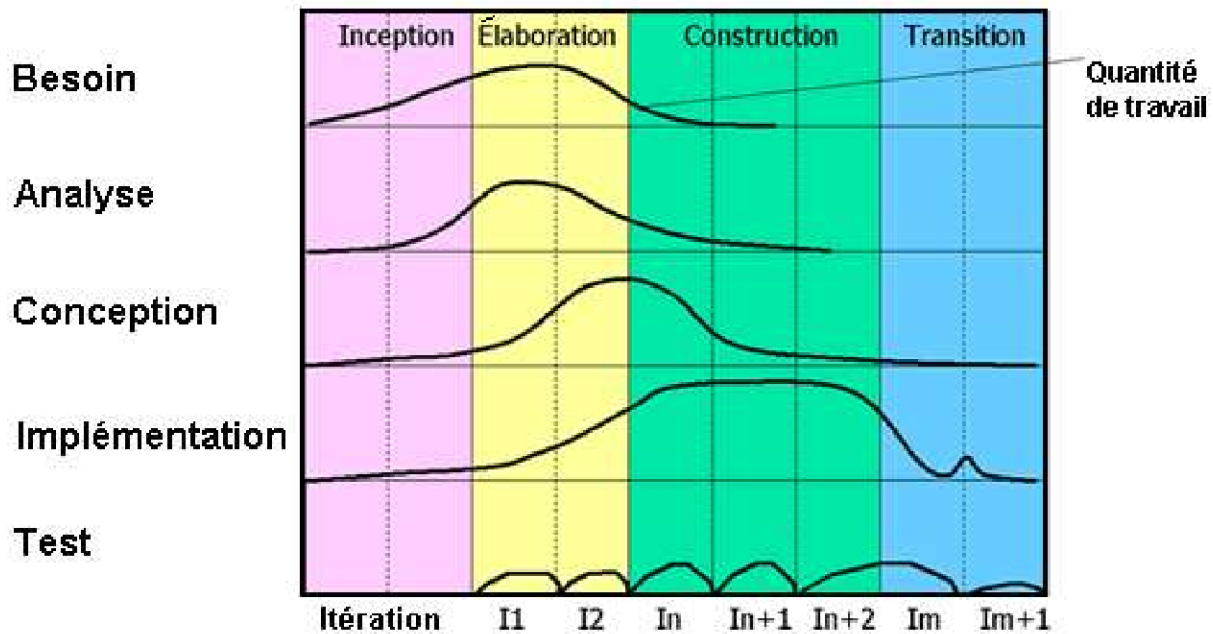
- Une vision commune pour l'équipe
- la satisfaction du client (Instaurer la confiance autrement et éviter les effets pervers d'un contrat)
- Découper le projet autrement
- par fonctionnalité
- Organiser en cycles de développement réduits
- Itérations (adaptabilité au réactives aux nouveaux besoins et réceptives aux nouvelles solutions)

## Méthode UP

Méthode UP : (méthode générique de développement de logiciel s'appuie sur UML)

- Itératif et incrémental : projet est découpé en itérations incrémentales pour réduire la complexité et de le bien maitriser ;
- Centré sur l'architecture : les différentes vues du système qui doit être construit; (forme)
- Piloté par les risques : limiter les risques majeurs des projets à fin déterminer l'ordre des itérations(plan, cout, délai... ) ;
- Guidé par les cas d'utilisation: L'architecture et les cas d'utilisation doivent évoluer de façon parallèle (fonction).

## Méthode UP



## Méthode UP

- Inception : Traduit une idée en vision de produit fini et présente une étude de rentabilité pour ce produit ;
- Elaboration : Permet de préciser la plupart des cas d'utilisation et de concevoir l'architecture du système ;
- Construction : moment où l'on construit le système, l'architecture de références se transforme en produit complet ;
- Transition : produit est en version bêta. essai du produit, détection des anomalies et défaut, formation des users.



Méthode agile XP : centrée sur le client et le code source.

- ◆ livrer très tôt une première version du logiciel ;
- ◆ Favoriser la collaboration entre les membres d'équipe ;
- ◆ Limiter les phases non-productive, adapter les changements.

Q1: Que doit faire le logiciel ?

Q2: Comment réaliser le logiciel ?

Q3: Quels sont les moyens pour réduire les durées de phases de modélisation?

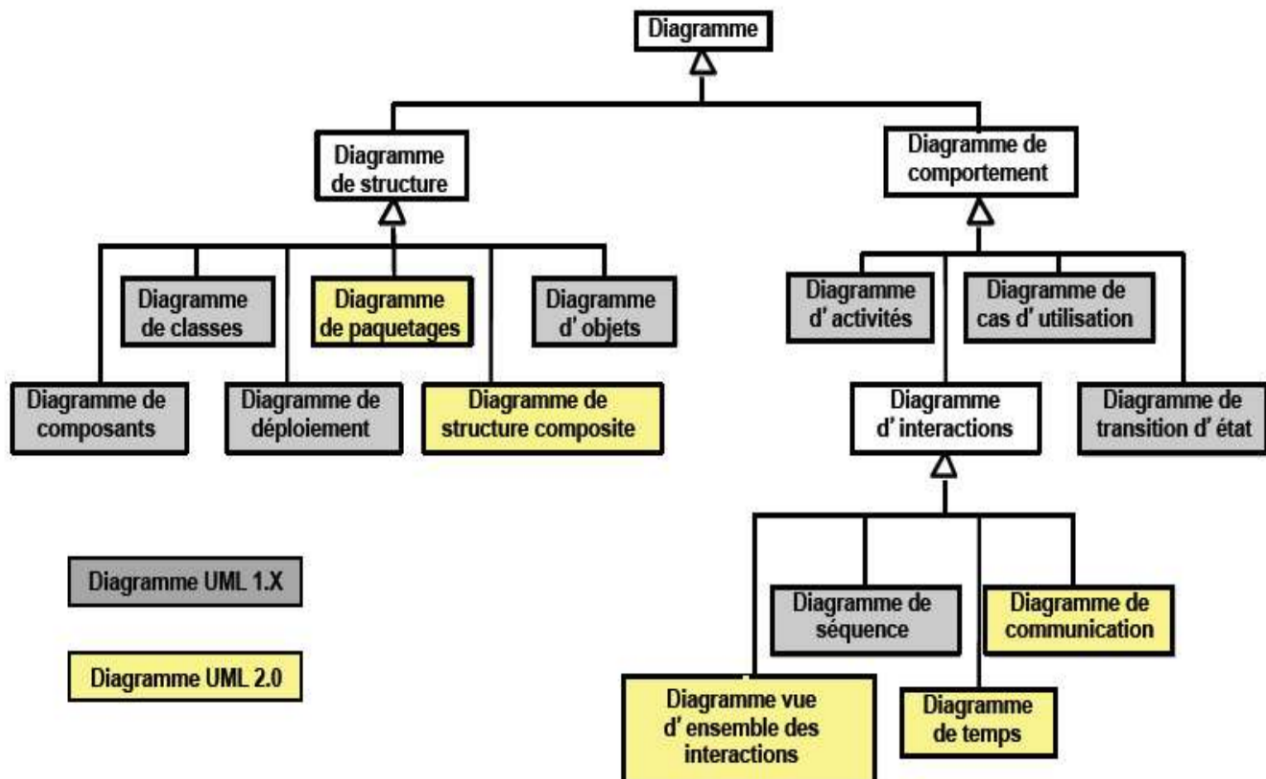
- ◆ UP répond bien au Q1 et Q2 : détermine les fonctionnalités à implémenter du logiciel, spécifie les phases à suivre pour réaliser un logiciel ;
- ◆ XP répond au Q3 : s'articule sur l'essentiel de modélisation, livre plus rapidement une version du logiciel ;
- ◆ XUP : réduire le temps de modélisation en gardant l'efficacité du projet.



UML(Unified Modeling Language ):

- Langage de modélisation d'un système orientée objet ;
- Représentation graphique établie par des diagrammes ;
- UML n'est pas une méthode, est une norme qui s'appuie sur des notations et des règles syntaxiques spécifiées par l'OMG (Object Management Group) ;

## Diagrammes UML



### Diagrammes statiques(structurels) :

- Diagramme de classe : définit les blocs de bases d'un modèle (classe, interface, types,...) ;
- Diagramme d'objet : montre comment les instances sont reliés et utilisées pendant l'exécution ;
- Diagramme de composant : décrit les éléments physiques et leurs relations dans l'environnement de réalisation, montre les choix de réalisation ;

### Diagrammes dynamiques(comportementaux)

- Diagramme de cas d'utilisation : décrit les besoins de l'utilisateur ;
- Diagramme d'activités : définit les flux de bases, points de décision et d'action d'un processus généralisé ;
- Diagramme d'état-transitions : exprime le comportement dynamique des objets(cycle de vie d'une classe) ;

### Diagrammes dynamiques (comportementaux)

- Diagrammes d'interactions :
- Diagramme de séquence : montre la séquence des messages échangés entre des objets utilisant une ligne de temps verticale ;
- Diagramme de communication : montre séquence de communication entre objets pendant l'exécution durant une instance de collaboration ;
- Diagramme d'interaction vue globale : fusionne diagramme d'activité et de séquence pour combiner les points de décisions et des flux ;
- Diagramme de temps : fusionne les diagrammes de séquences et d'états pour fournir une vue de l'état d'un objet dans le temps et les messages modifiant son état ;

**Les diagrammes les plus utilisés sont diagrammes de cas d'utilisation, classes, séquence, d'activités et d'état-transition.**

## Langage des contraintes d'objet (OCL)

### OCL permet de :

- Noter une condition ou une restriction sémantique exprimée grâce à un formalisme textuel simple ou composé ;
- Désigner une restriction qui doit être appliquée par une implémentation correcte ;

- Phase d'inception : recensement et définition des besoins ;
- Phase d'analyse ;
- Phase de conception ;
- Phase de réalisation.

### **Phase d'inception : recensement et définition des besoins :**

- Lister l'ensemble des exigences du client issues du cahier de charges ;
- Regrouper par intention d'acteur, puis diagramme de contexte ;
- Construire un diagramme de uses cases et faire la description de haut niveau.

### Phase d'analyse :

- Description détaillée pour chaque use case ;
- Diagramme de séquence par use case(diagramme de séquence boîte noire) ;
- Superposition des diagrammes de classes par use case.
- Contrat de LARMAN est réalisé pour chaque opération du système.
- Diagramme de séquence détaillé(Methodes) (diagramme de séquence boîte blanche ) ;

### Phase de conception :

- Réaliser le diagramme de collaboration à partir des diagrammes de classe et des contrats d'opération ;
- Établir les diagrammes d'état-transition des objets les plus complexe ;
- Réaliser le diagramme de classes de conception en appliquant les patterns de conception(GRASP patterns).

### Phase de réalisation :

- Déduire modèle logique de données (création de la BD) à partir du Diagramme de classe d'analyse ;
- Implémenter les objets métiers et leurs associations en se basant sur Diagramme de classe de conception et les contrats de LARMAN.

## Module : Génie logiciel

### Chapitre 3: Phase d'inception

Pr LAHCEN MOUMOUN  
[ensablearn@gmail.com](mailto:ensablearn@gmail.com)

Département Génie Informatique & Mathématiques

## Phase d'inception

- Nécessité de comprendre et structurer les besoins du client ;
- Spécification des interactions entre l'environnement externe et le système ;
- schématisation des besoins fonctionnels et opérationnels pour les valider par le client.



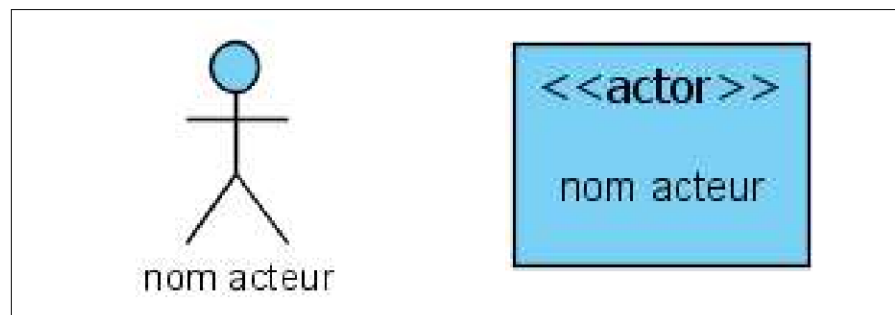
## Exigences

- Recensement des besoins à partir des cahiers de charge ou les enquêtes menés avec les utilisateurs ;
- Fonctions sont étiquetées par des numéros de référence.

Référence	Exigences	Référence	Exigences
R1	Démarrer d'un nœud source	R13	Laisser passer
R2	Changer de voie	R14	Arrêter la circulation
R3	Changer de route	R15	Changer feu de signalisation
R4	Tourner à droite	R16	Créer un profil
R5	Tourner à gauche	R17	Ecarter un conducteur
R6	Accélérer	R18	Résoudre conflit
R7	ralentir	R19	Paramétrer cycle
R8	S'arrêter	R20	Paramétrer phase
R9	Reprendre la circulation dans la simulation	R21	Définir un comportement au conducteur
R10	Passer une intersection	R22	Pénaliser un conducteur
R11	Traverser une route	R23	Editer synthèse
R12	Initialiser la simulation		

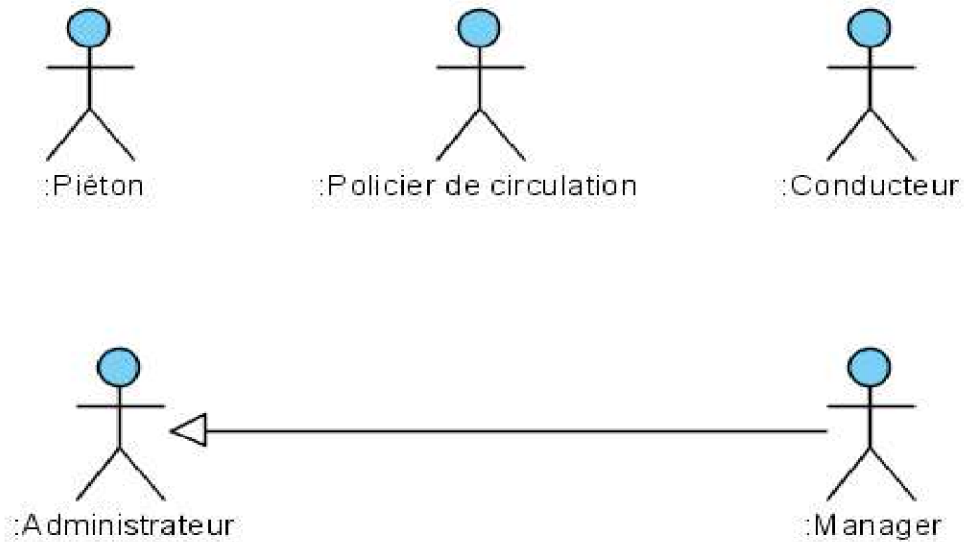
## Acteurs

- Un acteur est une entité externe qui interagit avec le système ;
  - Etre humain
  - Processus
  - Autres systèmes
- acteurs peuvent être classés en héritage.

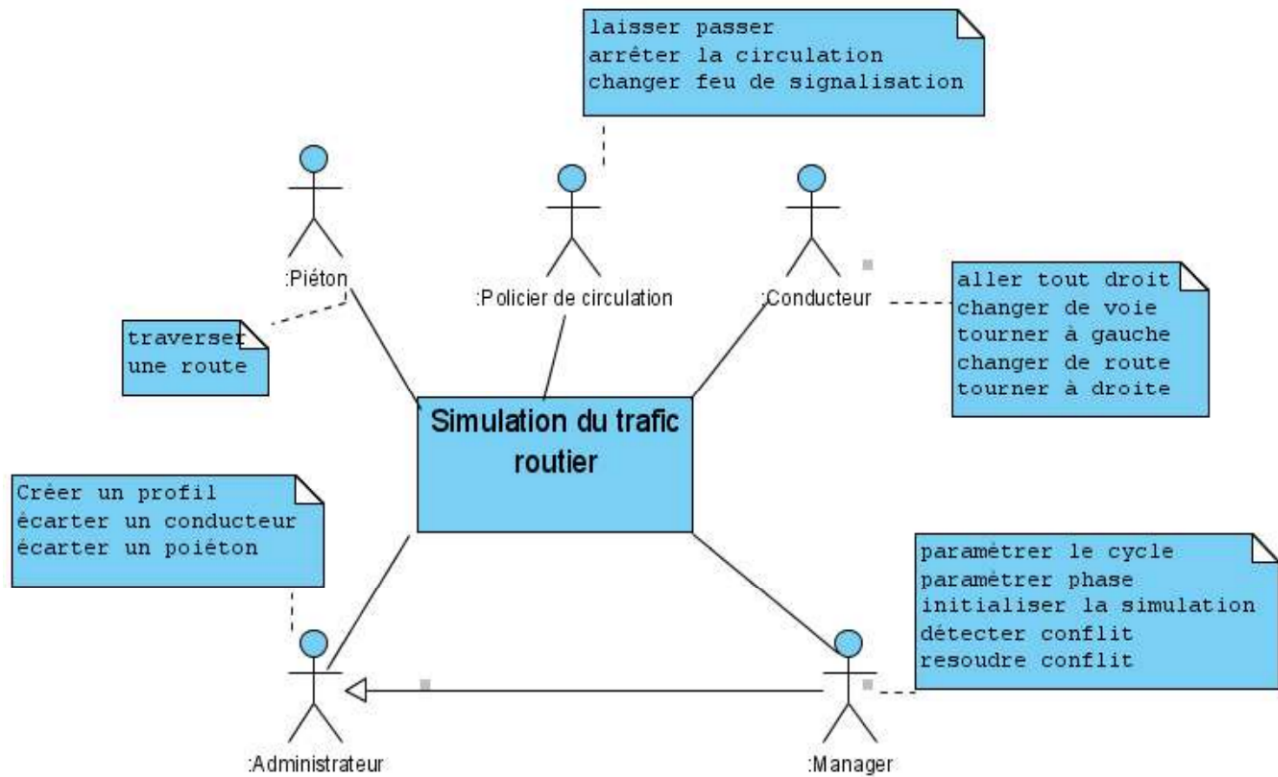


# Acteurs

Exemple d'acteurs du système :



# Diagramme de contexte statique



## Intentions d'acteurs

- Un enchainement de fonctions qui réalise un service complet à un acteur ;
- Cas d'utilisation fait l'objet d'une définition qui décrit l'intention de l'acteur du système.

Référence	fonctions	Intention d'acteur	Acteurs
R1	Démarrer à partir d'un nœud source	S'insérer dans une file	conducteur
R2	Changer de voie	Changer de voie	conducteur
R3	Changer de route	Changer de route	conducteur
R4	Tourner à droite	Changer de direction	conducteur
R5	Tourner à gauche	Changer de direction	conducteur
R6	Accélérer	Changer de vitesse	conducteur
R7	ralentir	Changer de vitesse	conducteur
R8	S'arrêter	Changer de vitesse	conducteur
R9	Reprendre la circulation dans la simulation	S'insérer dans une file	conducteur
R10	Passer une intersection	Passer une intersection	conducteur
R11	Traverser une route	Traverser une route	Piéton
R12	Initialiser la simulation	Initialiser simulation	Administrateur, Manager

## Intentions d'acteurs

Référence	fonctions	Intention d'acteur	Acteurs
R13	Laisser passer	Organiser la circulation	Policier
R14	Arrêter la circulation	Organiser la circulation	Policier
R15	Changer feu de signalisation	Organiser la circulation	Policier
R16	Créer un profil	Créer profil	Administrateur, Manager
R17	Ecarter un conducteur	Résoudre conflit	Administrateur, Manager
R18	Résoudre conflit	Résoudre conflit	Manager
R19	Paramétrer cycle	Paramétrer cycle	Administrateur, Manager
R20	paramétrer phase	paramétrer phase	Administrateur, Manager
R21	Définir un comportement au conducteur	Gérer comportement	Administrateur, Manager
R22	Pénaliser un conducteur	Gérer comportement	Administrateur, Manager
R23	Editer synthèse d'une simulation	Editer synthèse	Manager

## Intentions d'acteurs

Acteur principale obtient un résultat observable du système , par contre un acteur secondaire est sollicité par des informations complémentaires.



## Diagramme de cas d'utilisation

Use case permet :

- capturer le comportement du système, tel qu'un utilisateur extérieur le voit.
- scinder la fonctionnalité du système en unités cohérentes (cas d'utilisation) ayant un sens pour les acteurs.

Un cas d'utilisation

## Diagramme de cas d'utilisation

Association entre acteur et cas d'utilisation:

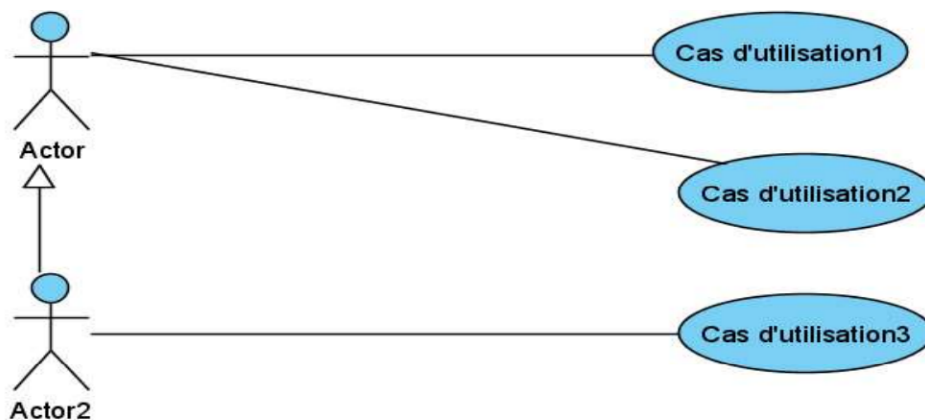
- Définit une interaction entre l'acteur et le système ;
- Une association peut porter une multiplicité.



## Diagramme de cas d'utilisation

Association entre acteurs :

- Ne peut être qu'une association de généralisation ;
- Signifie que l'acteur « fils » utilise les mêmes cas d'utilisation de l'acteur « parent ».



## Diagramme de cas d'utilisation

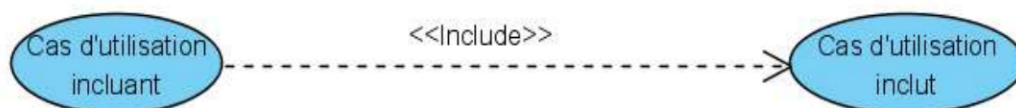
Types d'associations entre cas d'utilisation :

- L'inclusion ;
- L'extension ;
- L'héritage (généralisation/spécialisation).

## Diagramme de cas d'utilisation

### L'inclusion :

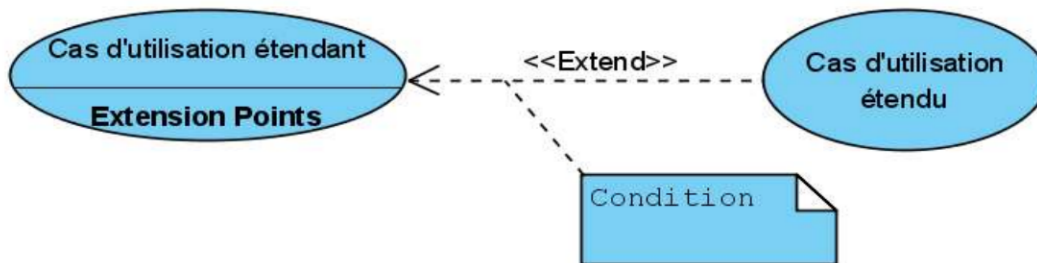
Un cas « A » inclut un autre cas d'utilisation «B», indique que le cas d'utilisation «A» contient obligatoirement le comportement défini dans «B».



## Diagramme de cas d'utilisation

### L'extension :

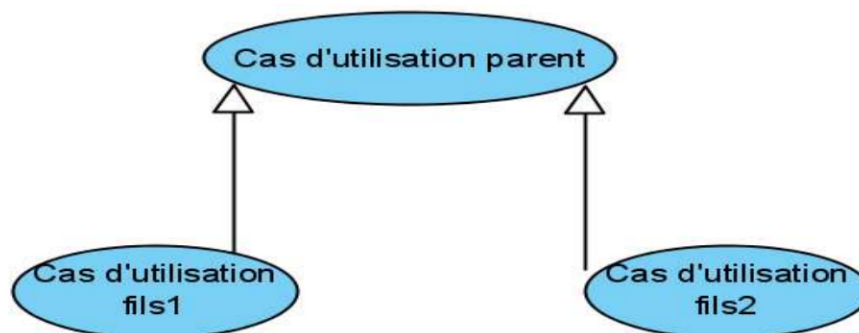
Un cas « A » étend un autre cas d'utilisation « B », signifie que le cas d'utilisation « B » peut être complétement à un certain point avec le comportement de « A » de manière facultatif.



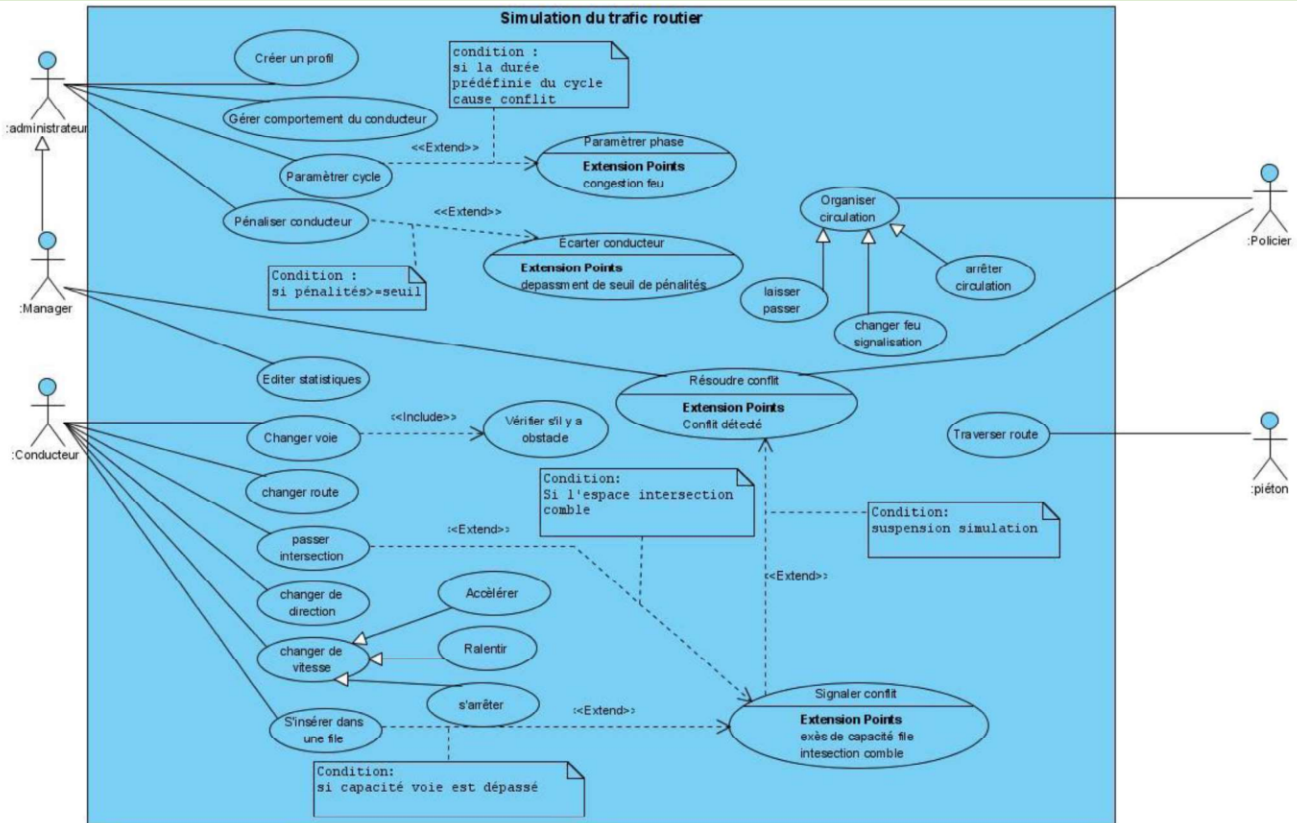
## Diagramme de cas d'utilisation

### L'héritage (généralisation/spécialisation) :

Cas d'utilisation fils est une spécialisation du cas d'utilisation parent.



# Diagramme de cas d'utilisation



# Description textuelle de haut niveau

Permet de :

- Décrire chaque use case, en précisant l'ensemble d'informations correspondantes ;
- Mettre en évidence l'aspect itératif adopté par la démarche UP-XP ;
- Représenter une introduction à la phase d'analyse, elle sera détaillée par une description textuelle de bas niveau.

Nom :	Utiliser un verbe à l'infinitif
Objectif :	Une description résumée permettant de comprendre l'intention principale du cas d'utilisation
Acteurs principaux :	Ceux qui vont réaliser le cas d'utilisation
Acteurs secondaires :	Ceux qui sont sollicités pour des informations complémentaires
L'évènement déclencheur :	L'évènement qui déclenche le cas d'utilisation
Dates :	Les dates de créations et de mise à jour de la description courante
Responsables :	Le nom des responsables
Version :	Le numéro de version
Terminaison :	Décrire la fin du cas d'utilisation



## Description textuelle de haut niveau

Description textuelle de haut niveau du cas d'utilisation : « Résoudre conflit d'insertion dans une file »

<b>Nom :</b>	<b>Résoudre conflit d'insertion dans une file</b>
Objectif :	Le manager résout le conflit en se basant sur les solutions possibles qui se présentent pour lui
Acteurs principaux :	Manager
Acteurs secondaires :	Policier
L'évènement déclencheur :	Un conflit est détecté et signalé au manager
Dates :	01/10/2010
Responsables :	Analyste
Version :	1.0
Terminaison :	Conflit disparaît

## Module : Génie logiciel

### Chapitre 4: Phase d'analyse

Pr LAHCEN MOUMOUN  
[ensablearn@gmail.com](mailto:ensablearn@gmail.com)

Département Génie Informatique & Mathématiques

### Phase d'analyse

- Nécessité d'analyser et de comprendre les fonctionnalités espérée par le client ;
- Présentation des séquences nominales et alternatives (erreurs et exceptions) ;
- Présentation des objets constituant système ;
- réalisation d'un contrat de LARMAN pour chaque opération.

## Description textuelle de bas niveau

Permet de :

- Etudier les scénarios du cas d'utilisation et les échanges entre les acteurs et le système ;
- Garder une vision métier sur le système ;
- Traiter les cas d'erreur et d'exception : point de déclenchement des exceptions dans le scénarios nominales ;
- Définir les références croisées des exigences, pré-condition, post-condition, description complète de use case, spécification matérielles et graphiques...

## Description textuelle de bas niveau

Description textuelle de bas niveau du cas d'utilisation : « Résoudre conflit d'insertion dans une file »

<b>Nom du use case :</b>	<b>Résoudre conflit d'insertion dans une file</b>
Acteur principal :	Manager
Acteurs secondaires :	Policier
Evénement déclencheur du use case	Un conflit est détecté et signalé au manager
Rôle du use case	Le manager résout le conflit en se basant sur les solutions possibles
Terminaison du use case	Conflit disparaît
Références croisées des exigences:	R17,R18
Pré-conditions :	La simulation est initialisée, des conducteurs sont connectés à la simulation
Post-conditions :	Le conflit est résolu et une synthèse de la solution est enregistrée

## Description textuelle de bas niveau

Présentation du déroulement du scénario nominale du cas d'utilisation: « Résoudre conflit d'insertion dans une file »

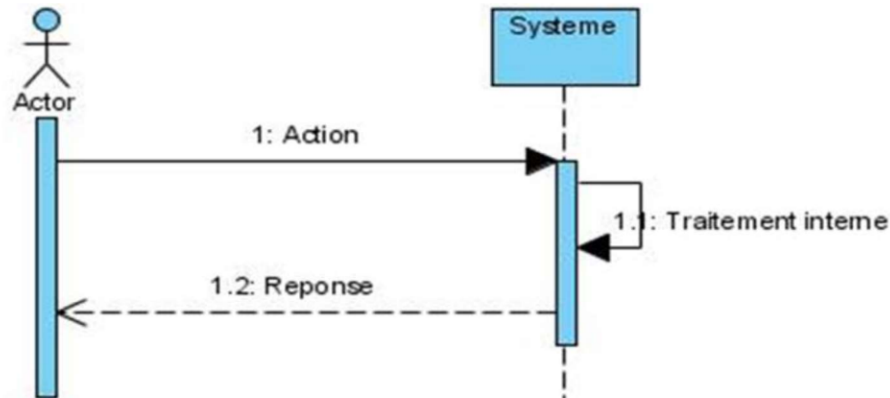
Actions des acteurs	Action du système
1)Le manager choisit l'option résoudre conflit	2)Le système affiche la palette des solutions
3)Le manager choisit l'une des possibilités de résolution de conflit à partir de la palette tant que ce dernier n'a pas disparu : -ajouter un feu de signalisation à un point de la section de la route -augmenter la taille de la route -Augmenter la vitesse réglementaire dans cette route -Changer dans les phases ou le cycle de feu de signalisation -Pénaliser un conducteur -Ajouter un policier de circulation	4)Le système teste la nature de la solution 5)Le système vérifie la disparition du conflit ou naissance d'autres conflit causés par l'application de la solution 6)Le système signale la fin de la solution
7)Le manager confirme la fin du conflit	8) Le système édite la synthèse de la résolution du conflit

## Description textuelle de bas niveau

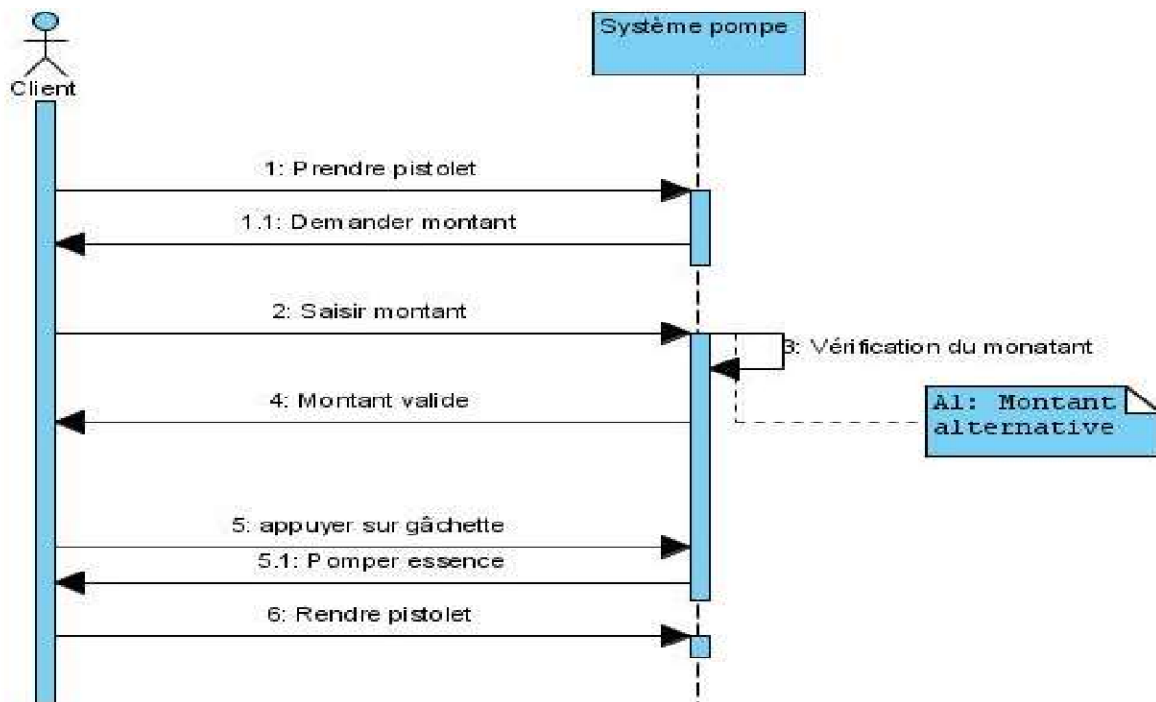
- Enchaînements alternatives :
  - A1 : Solution choisie par le manager est la délégation de la résolution du conflit au policier.
- Enchaînements d'exceptions :
  - E1: Solution défavorable(SD), solution conduit à un accident ou un autre conflit dans la même route ou une autre du réseau routier ;
  - E2 : Solution expirées(SE), expiration des solutions proposées par le système sans résoudre le conflit initial.
- À ce stade nous pouvons charger les infographistes par la réalisation des maquettes (interfaces graphiques) à fin de respecter la démarche UP-XP et présenter au client une vision prévisionnelle sur la réalisation du logiciel pour les approuver.

## Diagramme de séquence boîte noire

- Diagramme de séquence Système est dit aussi un diagramme de séquence boîte noire ;
- (Système = Boîte Noire), il décrit un scénario d'un cas d'étude. Un Diagramme de Séquence « Boîte noire » peut être enrichi par des actions internes (Souvent des vérifications) des notes de renvoi aux enchainements alternatifs et d'erreur.



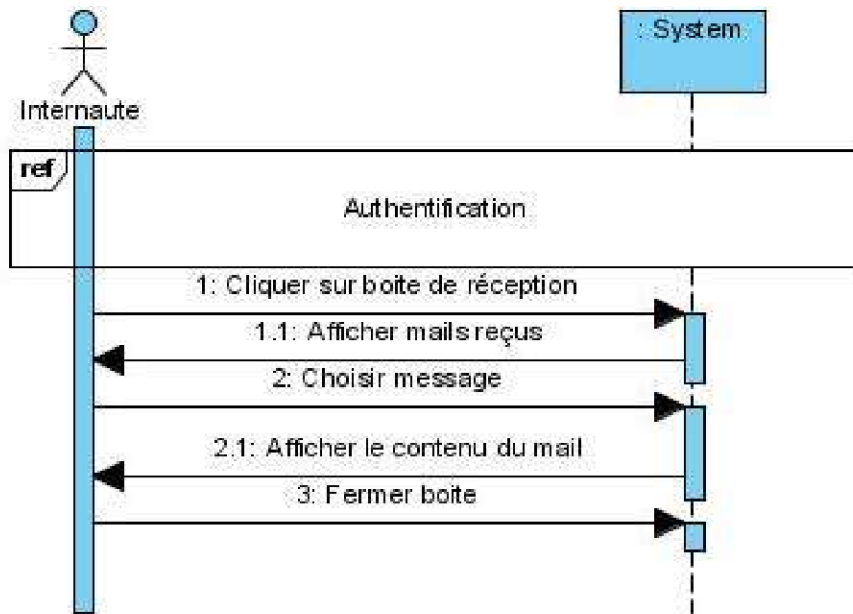
## Exemple de diagramme de séquence boîte noire



## Diagramme de séquence boîte noire

### Cadres d'interactions :

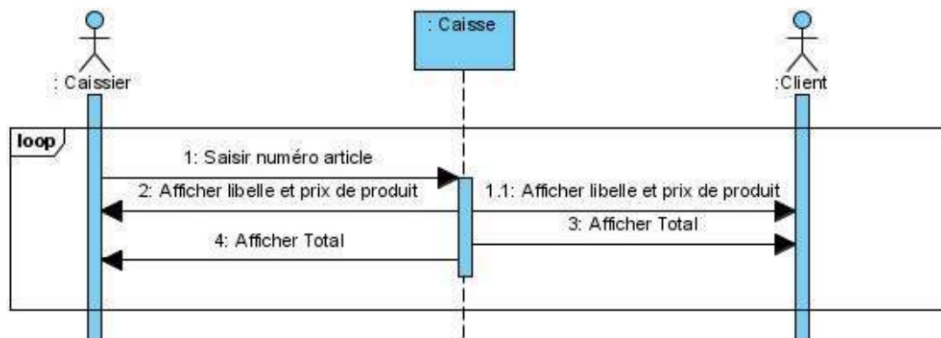
- REF : indique qu'un cas d'utilisation fait référence à une autre déjà étudié.



## Diagramme de séquence boîte noire

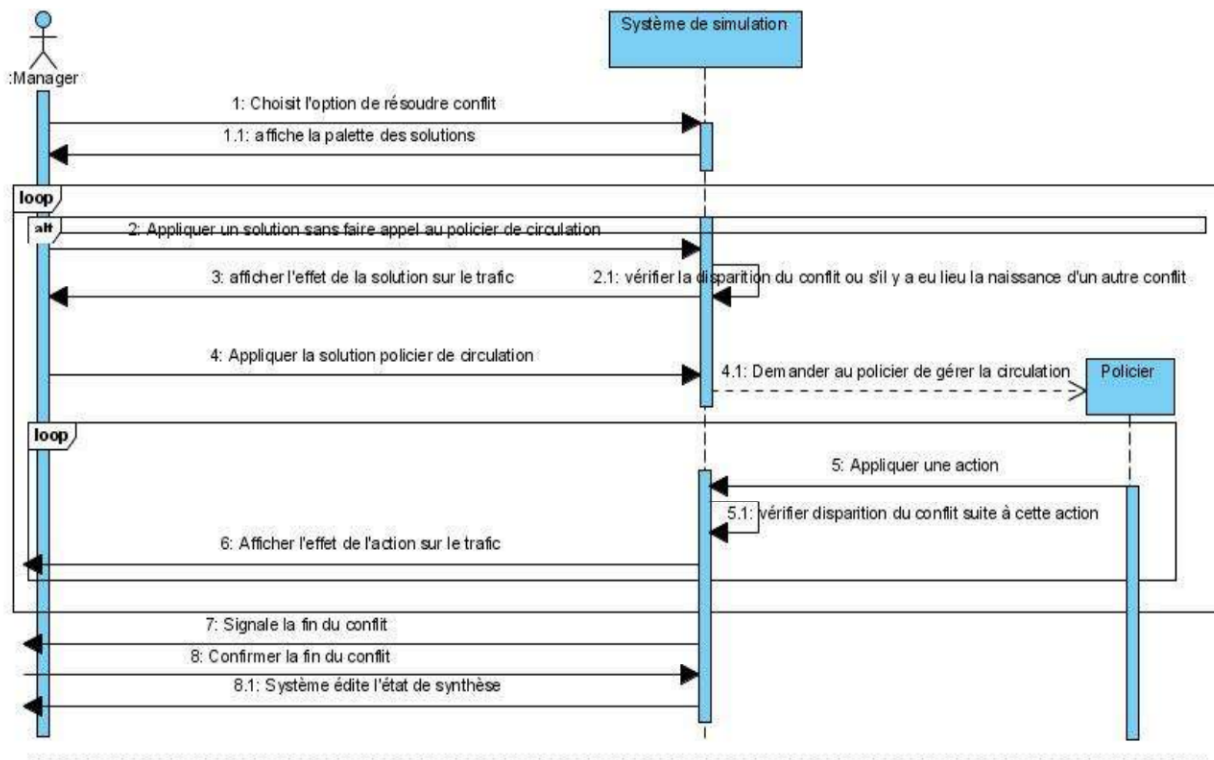
### Cadres d'interactions :

- LOOP : indique que les messages d'un cas d'utilisation échangés entre les acteurs et le système peuvent se répéter.



- OPT[Condition] : le contenu est exécuté si la condition est vraie.
- ALT : exécution à choix multiple

## Diagramme de séquence boîte noire



## Diagramme d'activité

- Il est opportun de construire un diagramme d'activité, si les diagrammes de séquence sont très nombreux et trop chargés ;
- Il décrit la dynamique d'un cas d'utilisation. En présentant les actions système, ils viennent d'illustrer et consolider la description textuelle des cas d'utilisation.



Début  
Système



Une activité du  
système



Condition

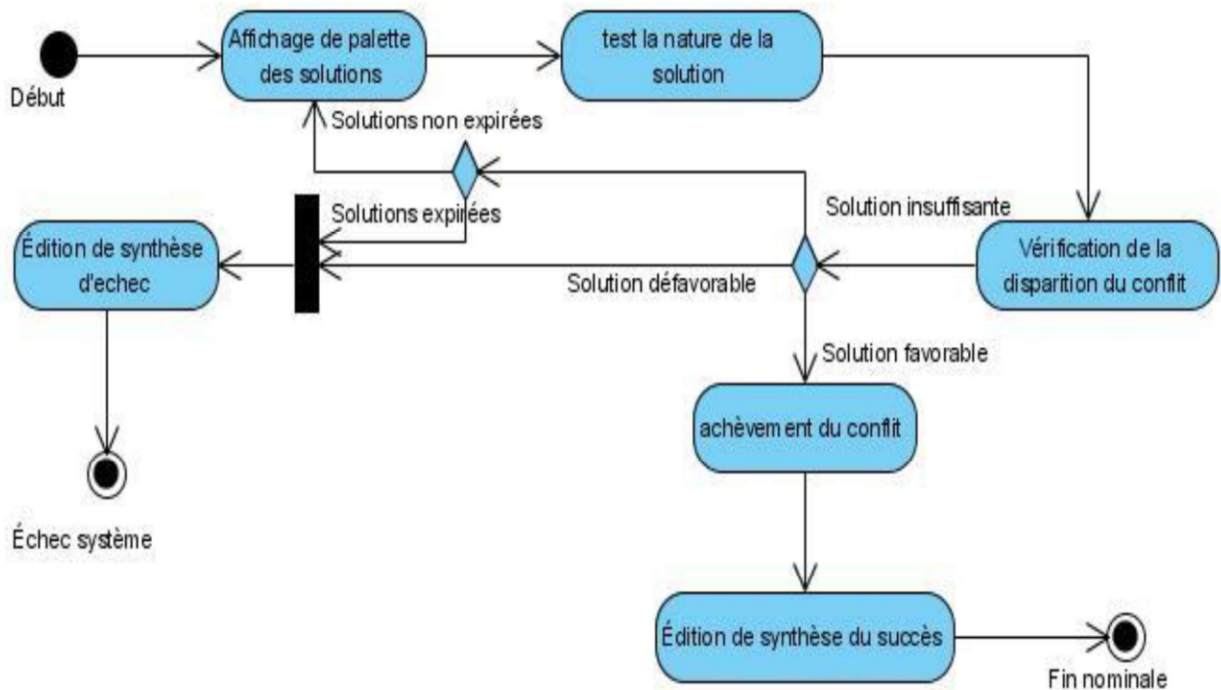


Fork ou join  
(séparer ou joindre)



Fin Système

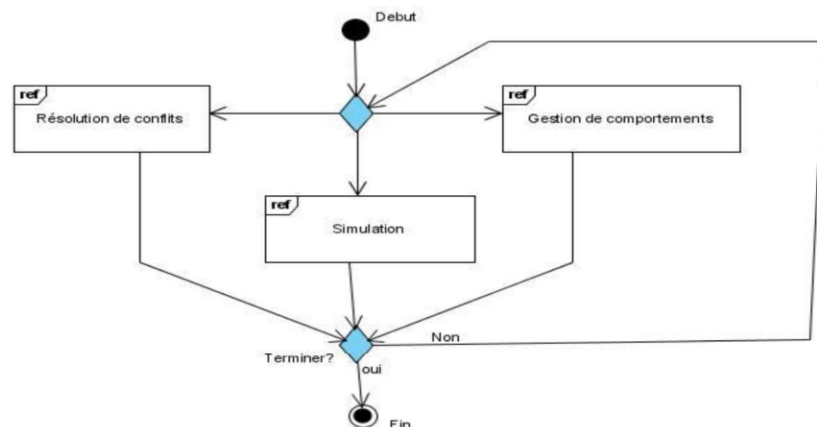
## Exemple de diagramme d'activité



## Diagramme d'interaction vue global

- Un Diagramme d'interaction vue global permet de représenter l'ensemble des transactions du système afin de donner une vue globale des fonctionnalités du Système.
- Ce diagramme regroupe les diagramme de séquence et d'activité.

Diagramme d'interaction vue global put être comme étant un menu principal de l'application





## Diagramme de classes d'analyse

- Dévoiler tous les objets métier qui constituent notre système (boite noire) qui vont interagir ensemble pour réaliser les cas d'utilisation ;
- Produire des modèles abstraits des objets du monde réel ;
- Dédurre les entités (objets) en se basant sur une analyse linguistique du cahier de charges ;
- Lier entre les objets par des associations d'appartenance ou de collaboration ;
- une vue statique car on ne tient pas compte du facteur temporel dans le comportement du système.

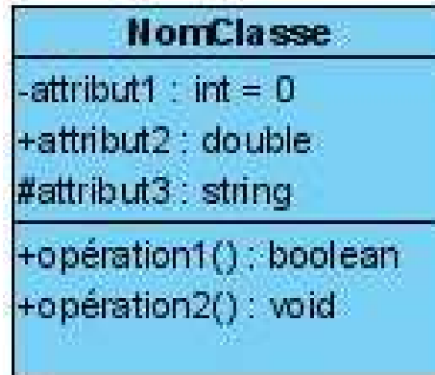
## Diagramme de classes d'analyse

- Issues de l'architecture **MVC (Model-View-Controller)**
- Classe présentation : GUI réalisée par une interface graphique ou une page web.
- Classe entité : stocke et manipule les données (logique métier).
- Classe contrôle : réalise la logique (contrôle) nécessaire pour interpréter les scénarios décrivant un cas d'utilisation.



## Diagramme de classes d'analyse

- Classe simple :



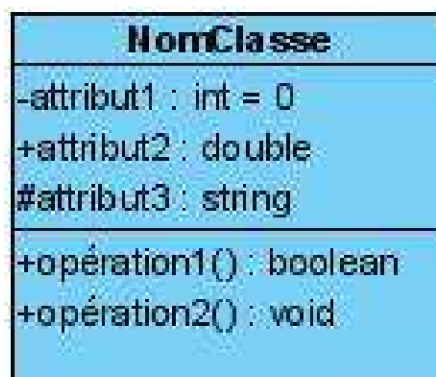
- Représentation graphique de l'encapsulation :

- + public()
- - private
- # protected
- / Attributs dérivés peuvent être calculés à partir d'autres attributs.

## Diagramme de classes d'analyse

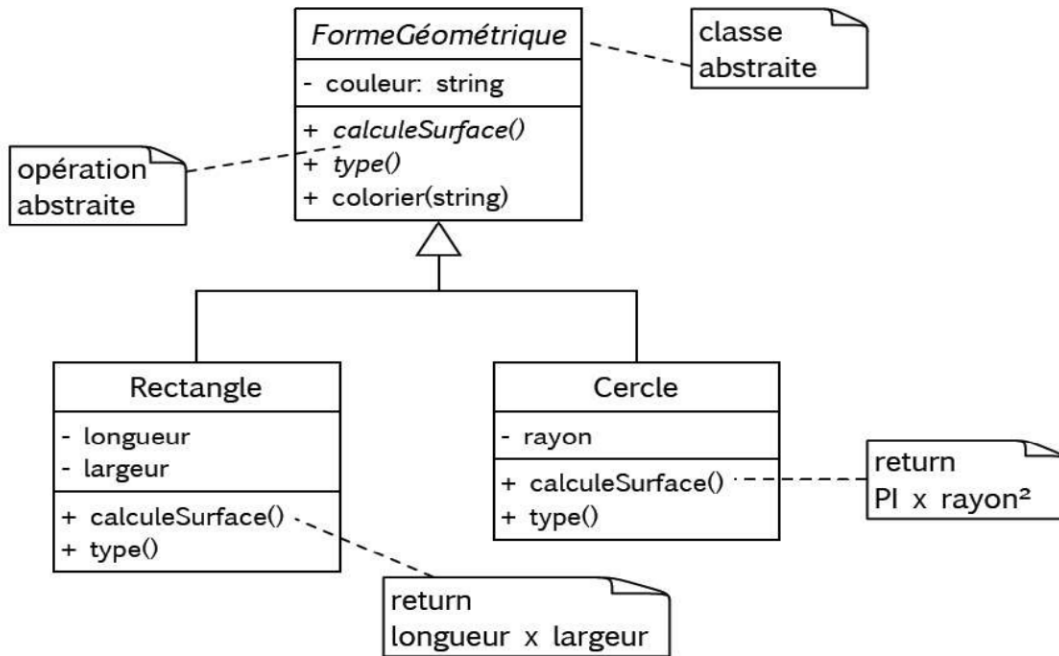
Classe abstraite : (non instanciable)

- une classe dont au moins l'une de ces méthodes n'a pas été implémentées. Elle sert avant tout à **factoriser du code**.



# Diagramme de classes d'analyse

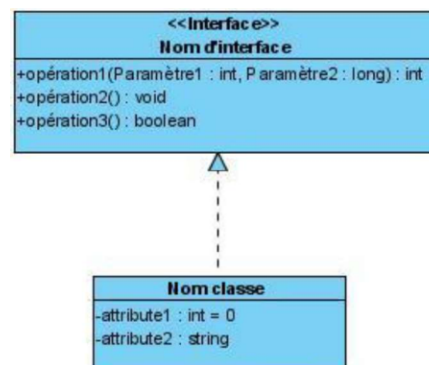
## Classe abstraite :



# Diagramme de classes d'analyse

## Interface :

- Une interface est équivalente à une classe abstraite dans laquelle aucune méthode ne serait implémentée ( les méthodes y sont seulement déclarées). Cela permet de définir un ensemble de **services visibles depuis l'extérieur**.

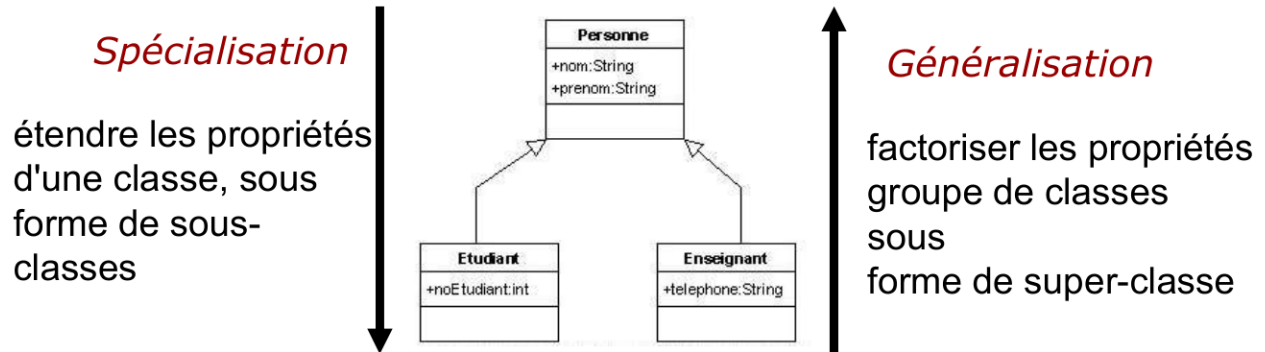


- Rq : les classes abstraites servent à factoriser du code, tandis que les interfaces servent à définir des contrats de service.

## Diagramme de classes d'analyse

### Héritage :

permet de créer une nouvelle classe à partir d'une classe existante . La classe dérivée contient les attributs et les méthodes de sa superclasse



- Chaque personne de l'université est identifiée par son nom, prénom
- Les étudiants ont plus un noEtudiant
- Les enseignants ont un numéro de téléphone interne

## Diagramme de classes d'analyse

### Association :

Connexion sémantique entre deux classes

### Navigabilité

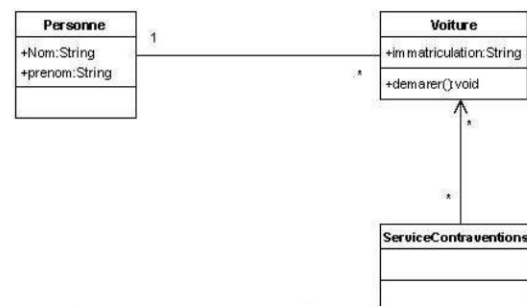
- Par défaut une association est navigable dans les deux sens



Chaque instance de voiture a un lien vers le propriétaire

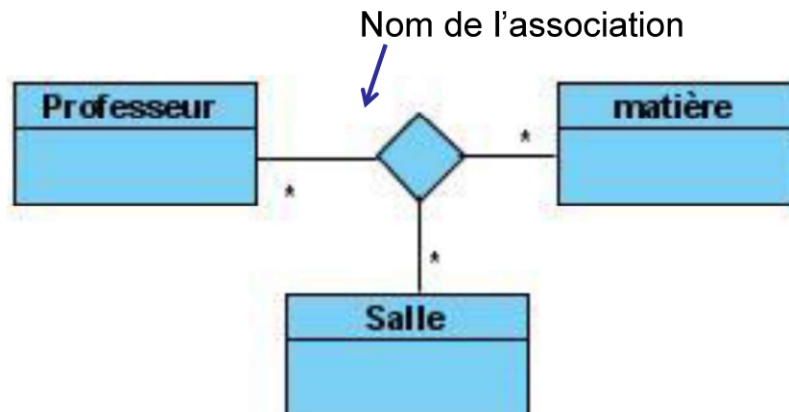
Chaque instance de Personne a un ensemble de lien vers les voitures

- Restriction de la navigabilité  
Le service de contravention est associé à une ou plusieurs voiture(s)  
La voiture ne connaît pas le service de contravention



## Diagramme de classes d'analyse

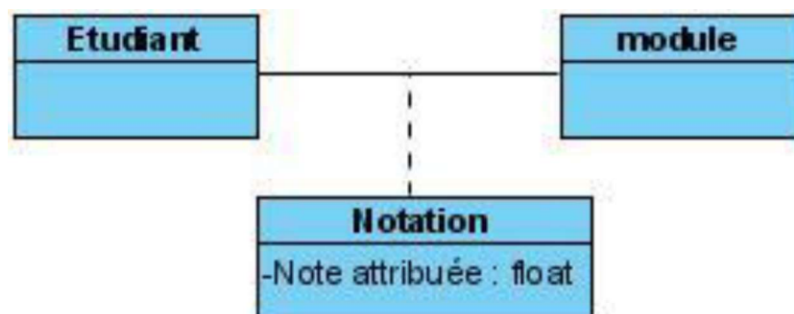
- Association n-aire :
- Type particulier d'association qui relie plus de deux classes ;



## Diagramme de classes d'analyse

### Classe Association :

- possède les caractéristiques des associations et des classes elle se connecte a deux ou plusieurs classes et possède également des attributs et des opérations ;
- Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

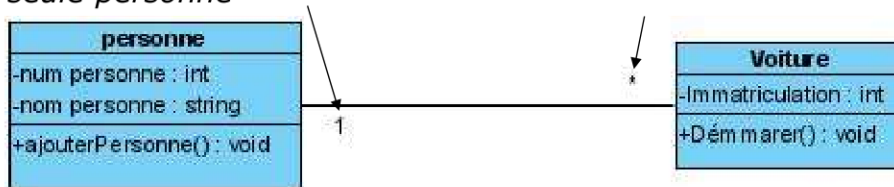


## Diagramme de classes d'analyse

Multiplicité	Signification
1 ou 1..1	Exactement 1
0..1	Zéro ou 1
N	Exactement N
N..M	Du minimum N au maximum M
0..* ou *	De zéro à plusieurs
1..*	De 1 à plusieurs
0..5,10	De 0 à 5 ou exactement 10

*Une voiture est achetée par une et une seule personne*

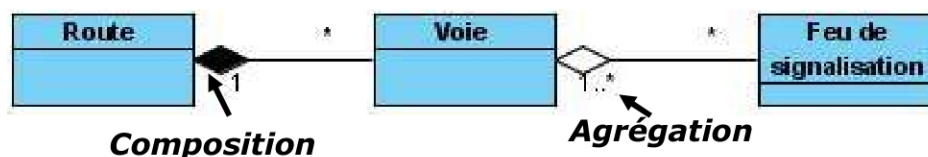
*Une personne peut acheter 0 ou n voitures*



## Diagramme de classes d'analyse

### Relations de contenance :

- Agrégation** : Lorsqu'un objet en contient d'autres, on parle d'agrégation. Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Graphiquement, on ajoute un losange vide du cote de l'agregat, comme indique a la figure. Elle n'entraîne pas non plus de contrainte sur la durée de vie des parties par rapport au tout.
- Composition** : La composition, également appelée agrégation forte, décrit une contenance Structurelle entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants. Une instance de la partie appartient toujours à au plus une instance de l'élément composite : la multiplicité du cote composite ne doit pas être supérieure a 1. Graphiquement, on ajoute un losange plein du cote de l'agregat.



- Elaboration de MLD permet de créer les tables de la base des données ;
- Détection des classes persistantes sont les classes d'une application qui implémentent les entités d'un problème métier ;
- Les règles de passage du diagramme de classe d'analyse à MLD sont les mêmes que MCD.

## Analyse Dynamique (LARMAN)

Opérations Système et contrat d'opérations : Pour Chaque cas d'utilisation, on dégage les opérations Système, et pour chaque opération on détermine son contrat, qui possède les informations suivantes :

- Nom d'opération
- Responsabilité
- Pré-conditions
- Post-conditions
- Exceptions et notes.

## Analyse Dynamique (LARMAN)

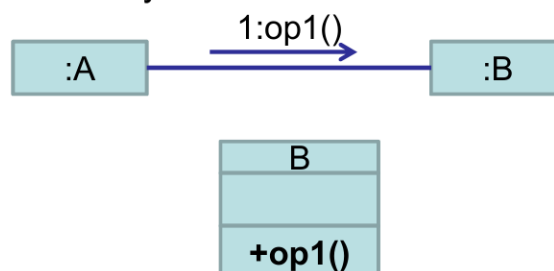
### Opérations Système et contrat d'opérations: Exemple

<b>Nom d'opération :</b>	<b>Insérer véhicule (type véhicule)</b>
Responsabilité :	Insérer un véhicule dans la file d'une route de type « type véhicule ».
Exigences :	R9 pour le use case s'insérer dans la file d'une route.
Pré-conditions	La route n'est pas encore saturée.
Post-conditions :	Un objet de type véhicule est crée et associée à une voie . Si l'insertion de ce véhicule cause un conflit , un objet de type conflit est crée et associé à la voie. Le manager est avisé pour appliquer une autre solution.
Exceptions et notes :	Si le type ne correspond pas à un type de véhicule prédéfini ou la route est saturée, un message d'erreur sera envoyé au manager et l'opération s'arrête.

## Analyse Dynamique (LARMAN)

### Opérations Système et contrat d'opérations

- Chaque opération Système va donner lieu a une étude dynamique sous la forme d'un diagramme d'interaction(Communication, Séquence boîte blanche ou collaboration) ;
- Les diagrammes d'interaction ainsi réalisés vont permettre d'élaborer le diagramme de classes de conception, et ceci en ajoutant principalement les informations suivantes aux classes issues du modèle d'analyse.

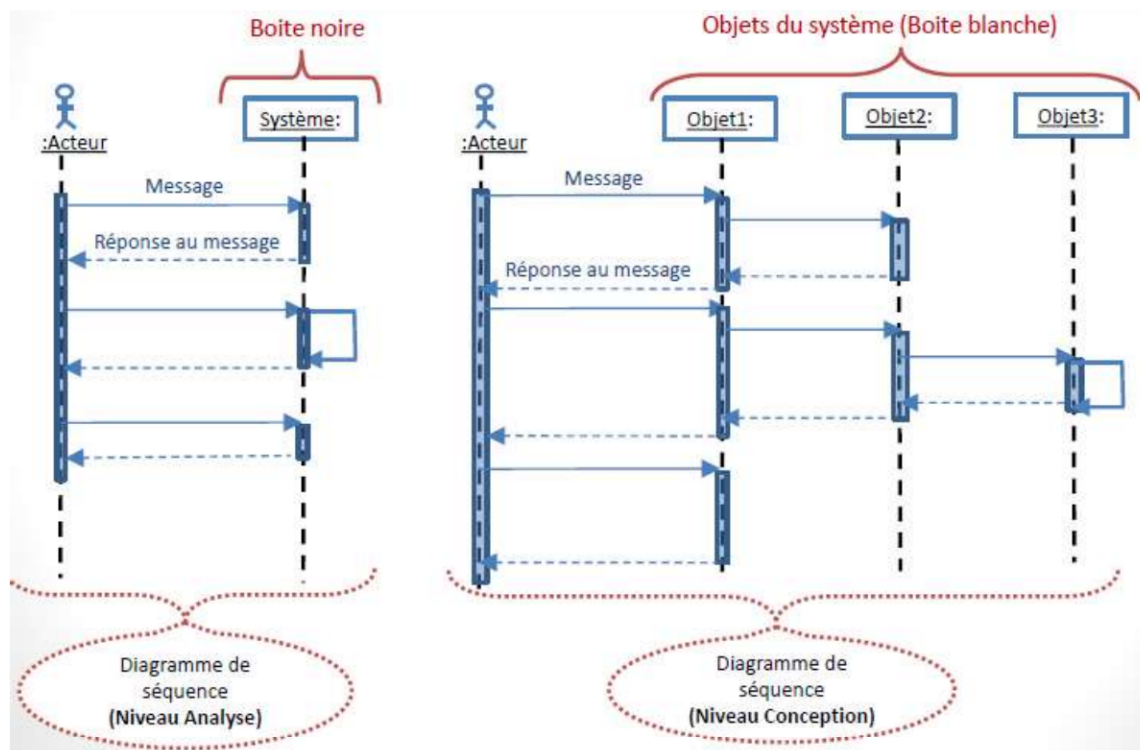




## Diagramme de séquence d'analyse boîte blanche

- Met l'accent sur la chronologie de l'envoi des messages entre les classes d'analyse et les acteurs principaux et secondaires.
- Dévoile les composants du système représenté ultérieurement dans un diagramme de séquence boîte noire. Le système boîte blanche se compose des objets présentatifs, logiques et métiers (Classes d'analyses).
- Représente les messages échangés entre les lignes de vie des acteurs principaux, acteurs Secondaires et classes d'analyse qui composent le système présentes dans un ordre chronologique temporel, le temps s'écoule de haut en bas.





## Diagramme de séquence d'analyse boîte blanche



## Diagramme de séquence d'analyse boîte blanche

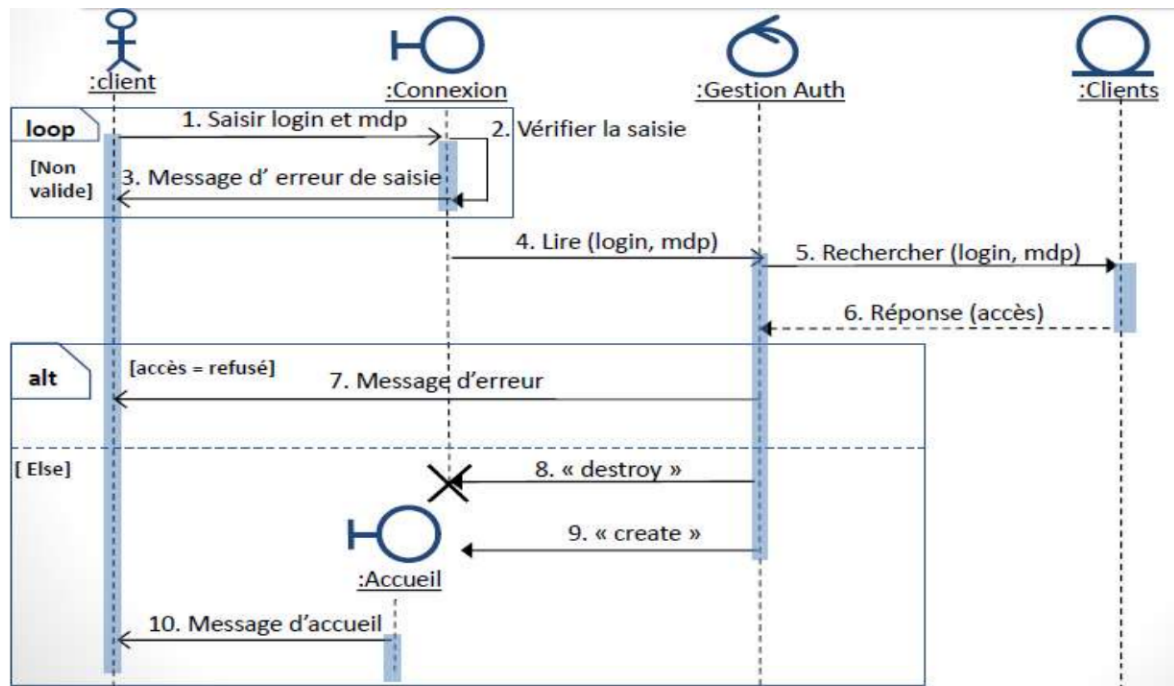
«Destroy» et « Create » sont deux stéréotypes de messages.

Il existe différents types de messages :

- Message **synchrone** 
  - Représenté par une flèche pleine ou complète et signifie que l'objet expéditeur envoie le message et reste bloqué tant que le destinataire n'a pas fini de traiter le message reçu.
- Message **asynchrone** 
  - Représenté par une flèche vide ou incomplète et signifie que l'objet expéditeur envoie le message et ne reste pas bloqué pendant le traitement du message par le destinataire.
- Message **réflexif** 
  - L'objet s'envoie un message à lui-même. L'expéditeur est lui-même le destinataire.
- Message **de retour** 
  - Représenté en pointillés. Le récepteur d'un message synchrone rend la main à l'émetteur du message en lui envoyant un message de retour

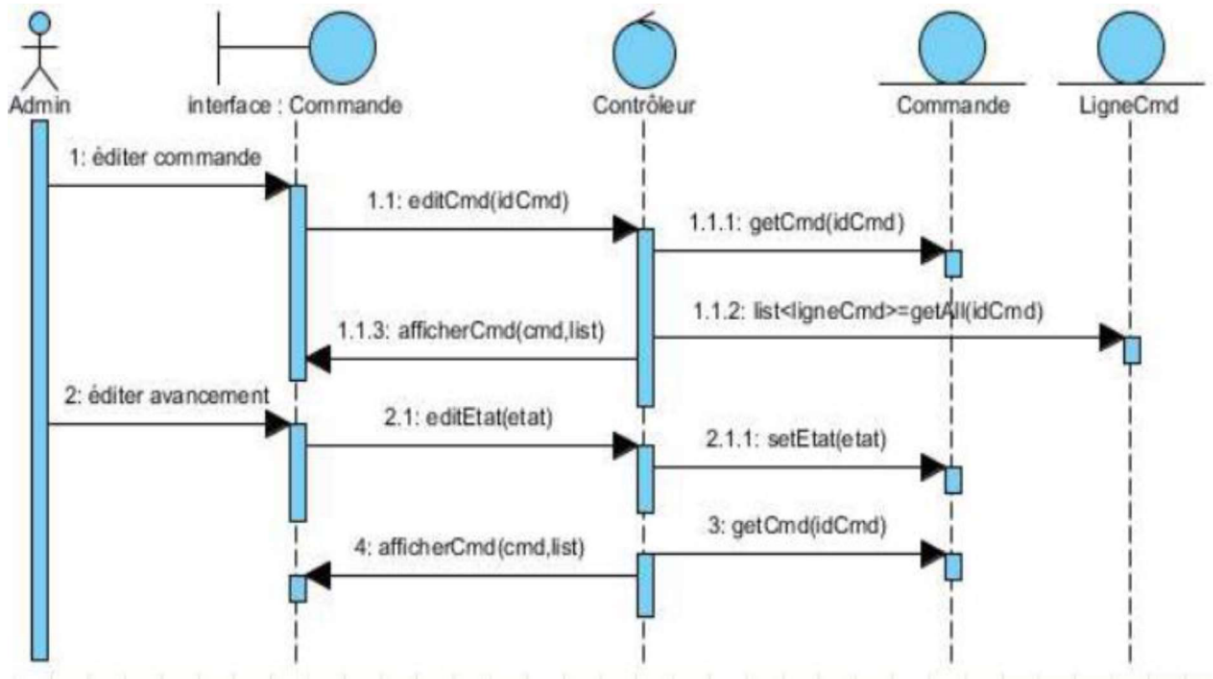
## Diagramme de séquence d'analyse boîte blanche

Séquence d'actions pour réaliser le UC : Authentification



## Diagramme de séquence d'analyse boîte blanche

Séquence d'actions pour réaliser le UC : Editer Commande



## Diagramme de séquence d'analyse boîte blanche

Séquence d'actions pour réaliser le UC : Ajouter produit au panier

