

# Programmation – Concepts avancées

- Théorie

- Analyse du concept objet
  - Classes
  - Encapsulation
  - Spécialisation
  - Polymorphisme
  - ....
- Modélisation de ce concept

## Un objet c'est quoi ?

- Entité identifiable du monde réel
  - Existence physique (cheval, livre) ou pas (un texte de loi)
  - Identifiable = désignable
- En UML :
  - Attributs
  - Méthodes
- En UML : objet toujours perçu dynamique
- Objets interagissent entre eux

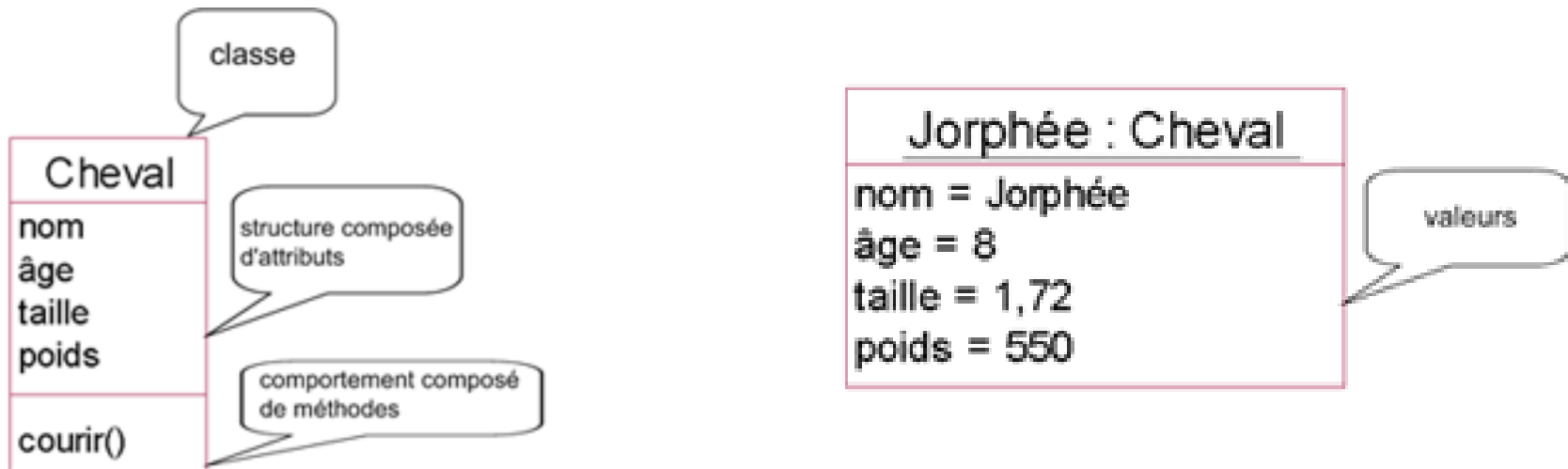
- Principe IMPORTANT
  - Retient juste les propriétés pertinentes

Exemple :

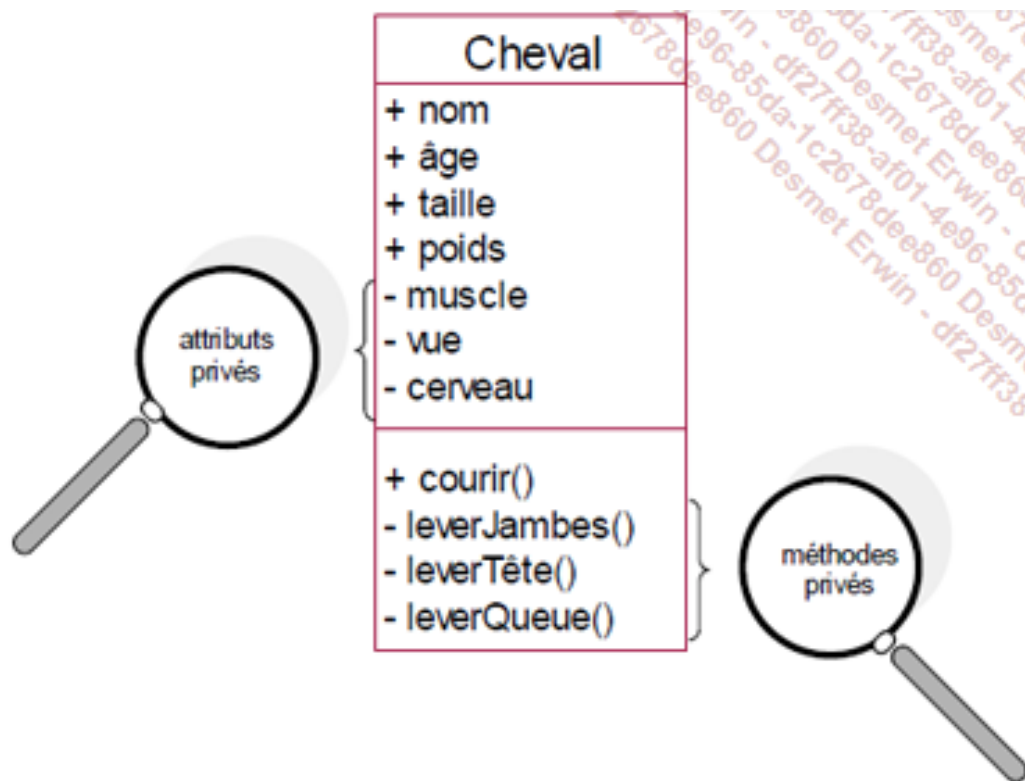
- *On s'intéresse aux chevaux pour l'activité de course. Les propriétés d'aptitude de vitesse, d'âge et d'équilibre mental ainsi que l'élevage d'origine sont pertinentes pour cette activité et sont retenues.*
- *On s'intéresse aux chevaux pour l'activité de trait. Les propriétés d'âge, de taille, de force et de corpulence sont pertinentes pour cette activité et sont retenues.*

## Les classes d'objets

- Un ensemble d'objets qui possède une même structure et un même comportement
- Un objet de classe est appelé instance de celle-ci.
  - Identité propre
  - Possède valeurs spécifiques pour ses attributs

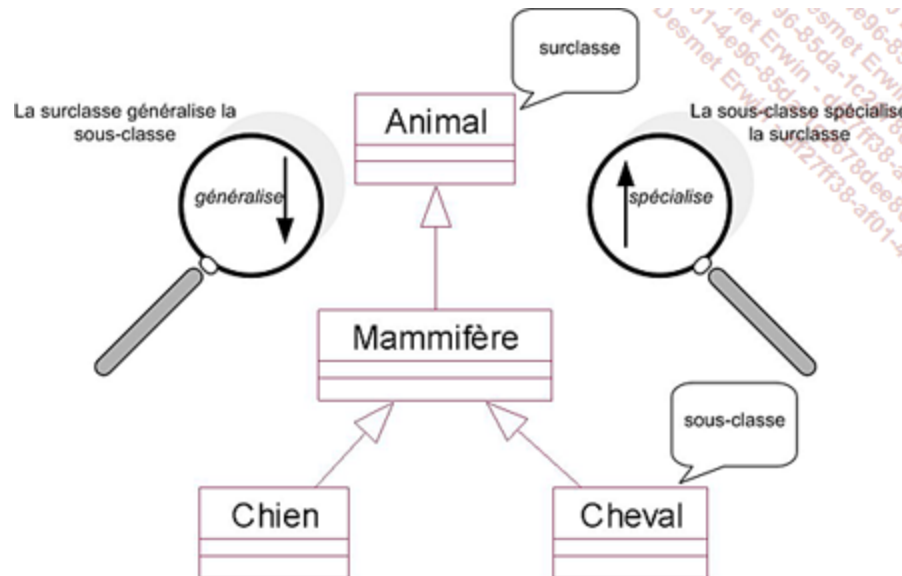


- Masquage des attributs et méthodes
- On parle de propriétés privées de l'objet
- Encapsulation = abstraction → Garde les publiques
- Niveau de la classe
- En UML : privés = - et publiques = +



# Spécialisation et Généralisation

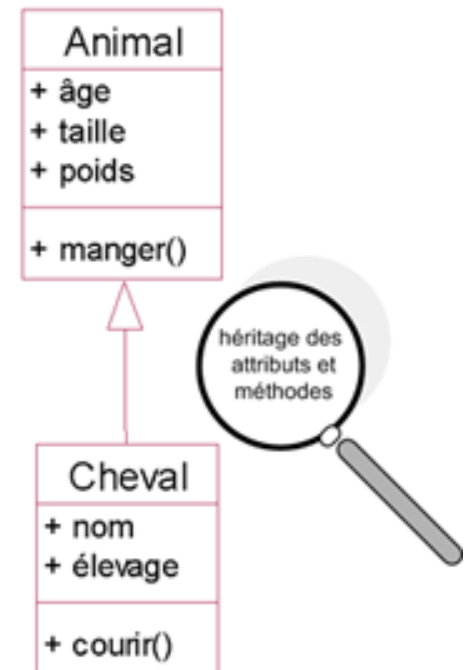
- Introduction différente
- Sous-ensemble
- On parle de sous-classe



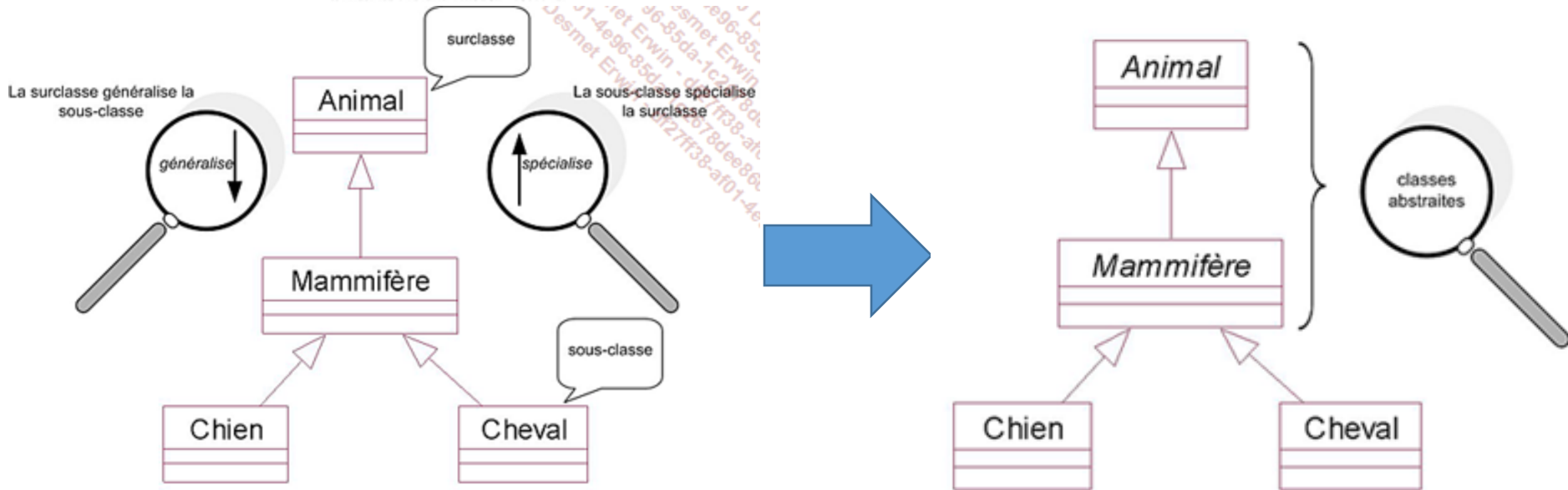


# L'héritage

- Donner à la sous-classe la structure et le comportement de la classe parent
- L'instance est aussi une instance de la surclasse
- Donc struct + comportement surclasse ET struct + comportement de la sous-classe
- Héritage = conséquence de spécialisation



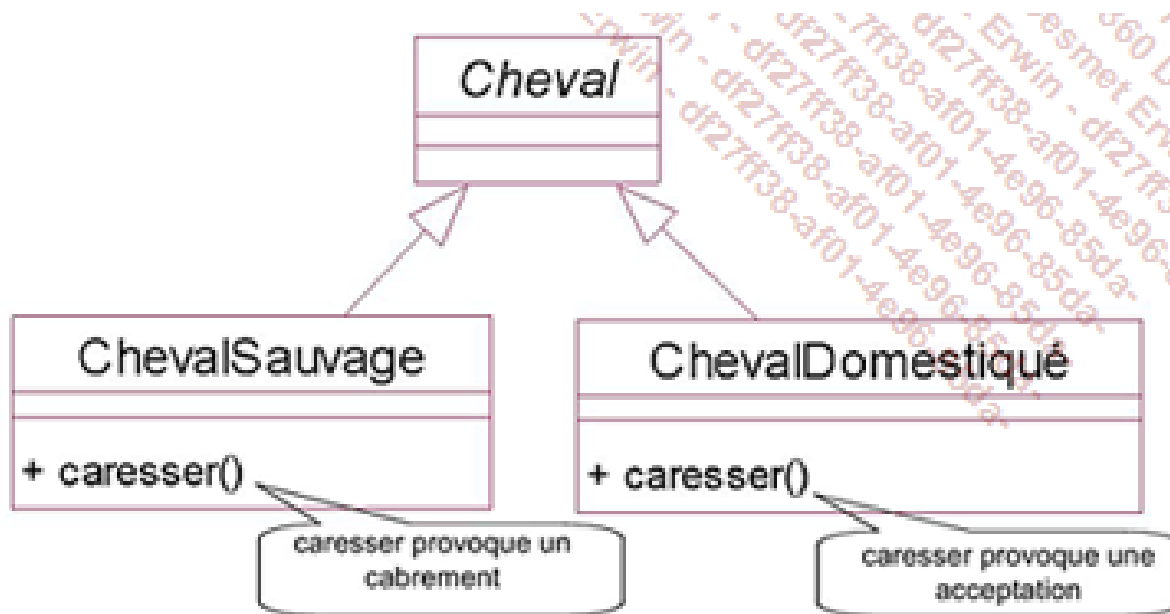
# Classes Abstraites et concrètes



- Classes avec instance → Cheval, Chien → Concrètes
- Classes sans instance → Animal → Abstraites
- Abstraites = avoir des sous-classes concrètes

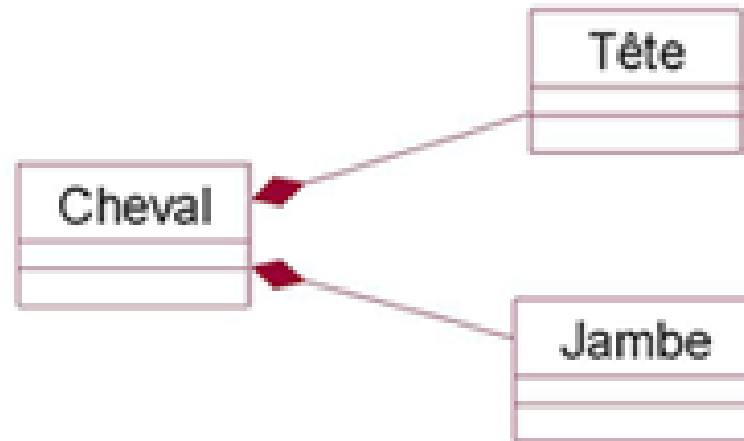
# Polymorphisme

- Classe(souvent abstraites) = ensemble d'objets différents
  - Instances de sous-classes différentes
  - Appel méthode même nom = effet différent



# Composition

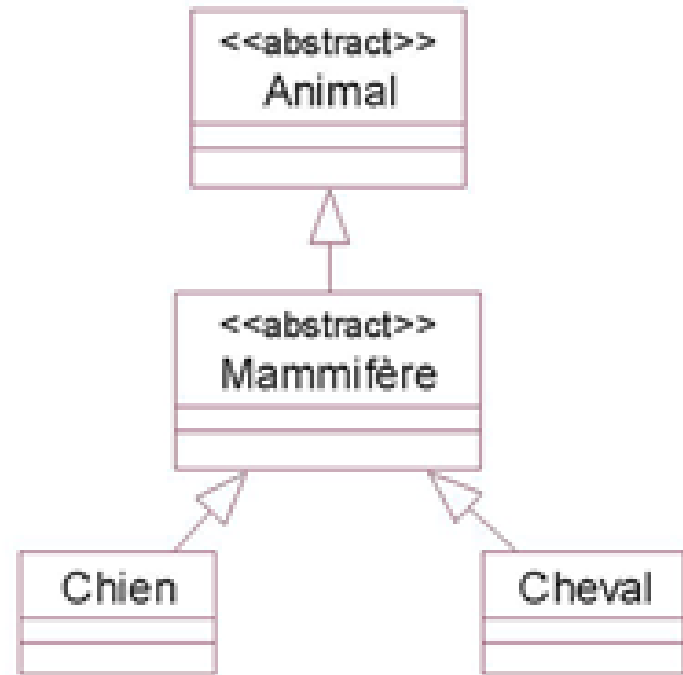
- Objet composé d'autres objets
- Définition sur la classe mais les liens entre les instances
- On les appelle composants



- 23 ●●●●



- Stéréotypes à pour but de spécialiser un concept
- Constitution : Mot clé entre guillemets
- Indépendant du système à modéliser
- On peut aussi simplement placer le nom de la classe en italique



# Petit Quizz





- Les cas d'utilisation :
  - Liste d'actions et d'interactions
  - Limites du systèmes

***Entre un acteur et le système, un cas d'utilisation décrit les actions et interactions liées à un objectif fonctionnel de l'acteur.***

- *Exemple : Considérons comme système un élevage de chevaux. L'achat d'un cheval par un client constitue un cas d'utilisation.*

- Les acteurs
  - Rôle d'un utilisateur externe
  - Définition étendue aux autres systèmes
- Deux catégories d'acteurs doivent être distinguées :
  - Les acteurs primaires, pour lesquels l'objectif du cas d'utilisation est essentiel et constitue un objectif de l'acteur.
  - Les acteurs secondaires, pour lesquels l'objectif du cas d'utilisation n'est pas essentiel bien qu'ils interagissent avec lui.
- Exemple : *Reprenons l'exemple précédent du cas d'utilisation de l'achat d'un cheval par un client. L'acheteur d'un cheval est un acteur primaire. Les haras nationaux qui enregistrent le certificat de vente constituent un acteur secondaire.*

- Les scénarios
  - Instance d'un cas d'utilisation
  - Un cas = plusieurs instances

***Comme une classe qui détient les aspects communs de ses instances, un cas d'utilisation décrit de façon commune l'ensemble de ses scénarios en utilisant des branchements conditionnels pour représenter les différentes alternatives.***

- Exemple : *L'achat de Sun par Florence constitue un exemple de scénario du cas d'utilisation d'achat d'un cheval. Toutes les alternatives du déroulement sont connues, car Florence a acquis Sun.*

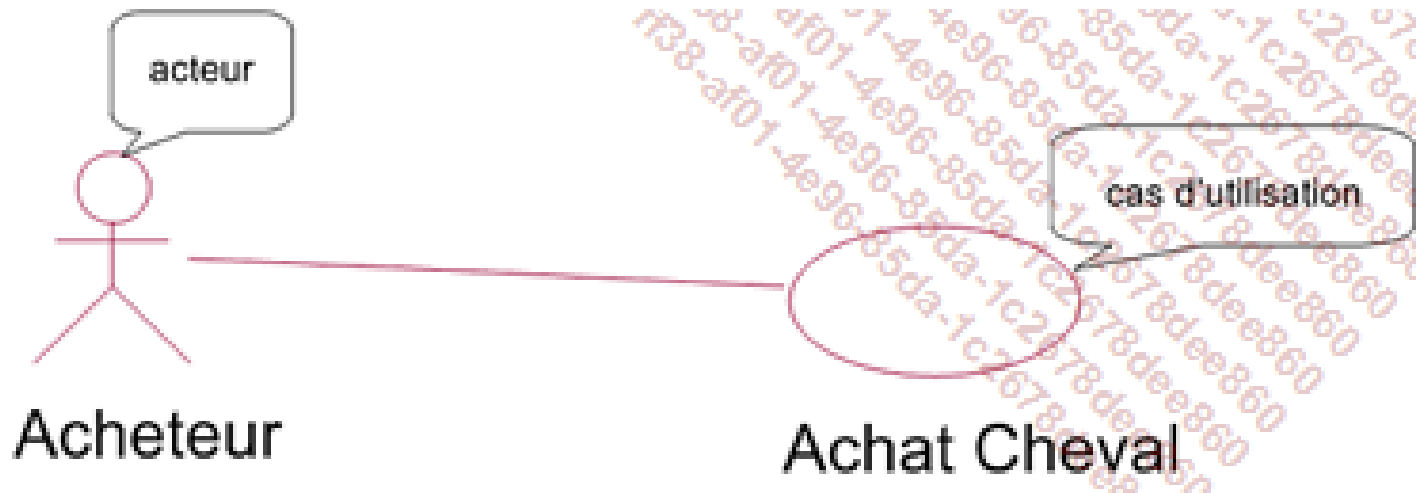
- Association entre acteurs et cas d'utilisation
  - Acteur → interaction définie par le cas

- Graphiquement = un trait

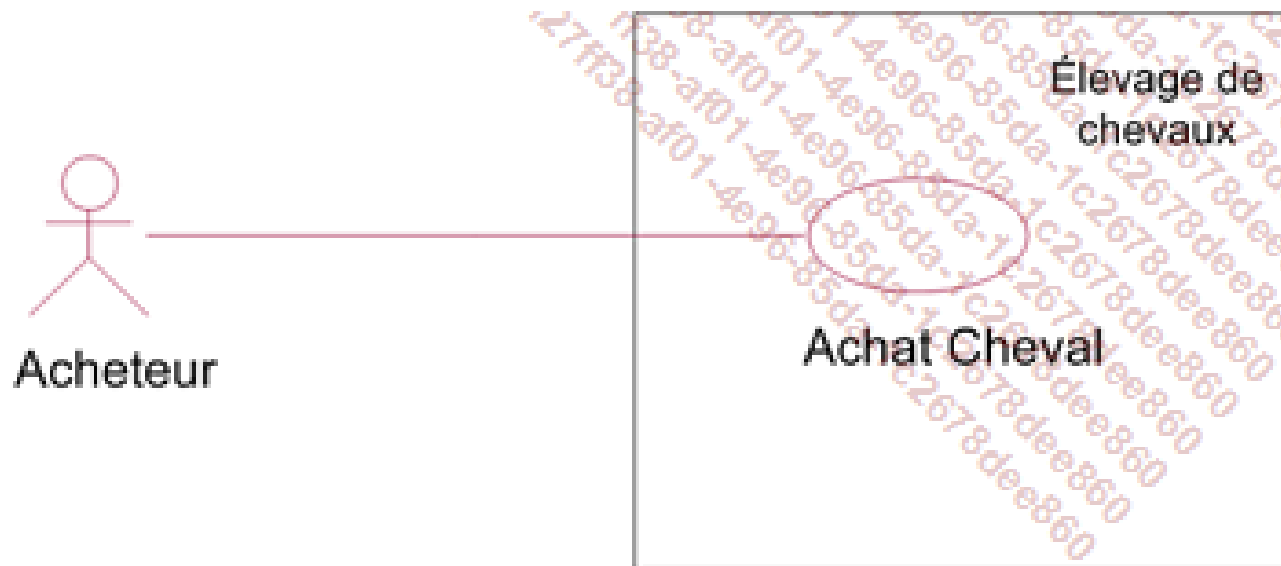


- On peut orienter avec une flèche (allant du demandeur vers receveur)
- *Exemple : L'association qui lie l'acteur **Acheteur** au cas d'utilisation **Achat Cheval** indique que cet acteur détient la capacité d'acheter un cheval en interagissant avec ce cas d'utilisation.*

- Cas simple : Ovale = Cas , Personnage = Acteur

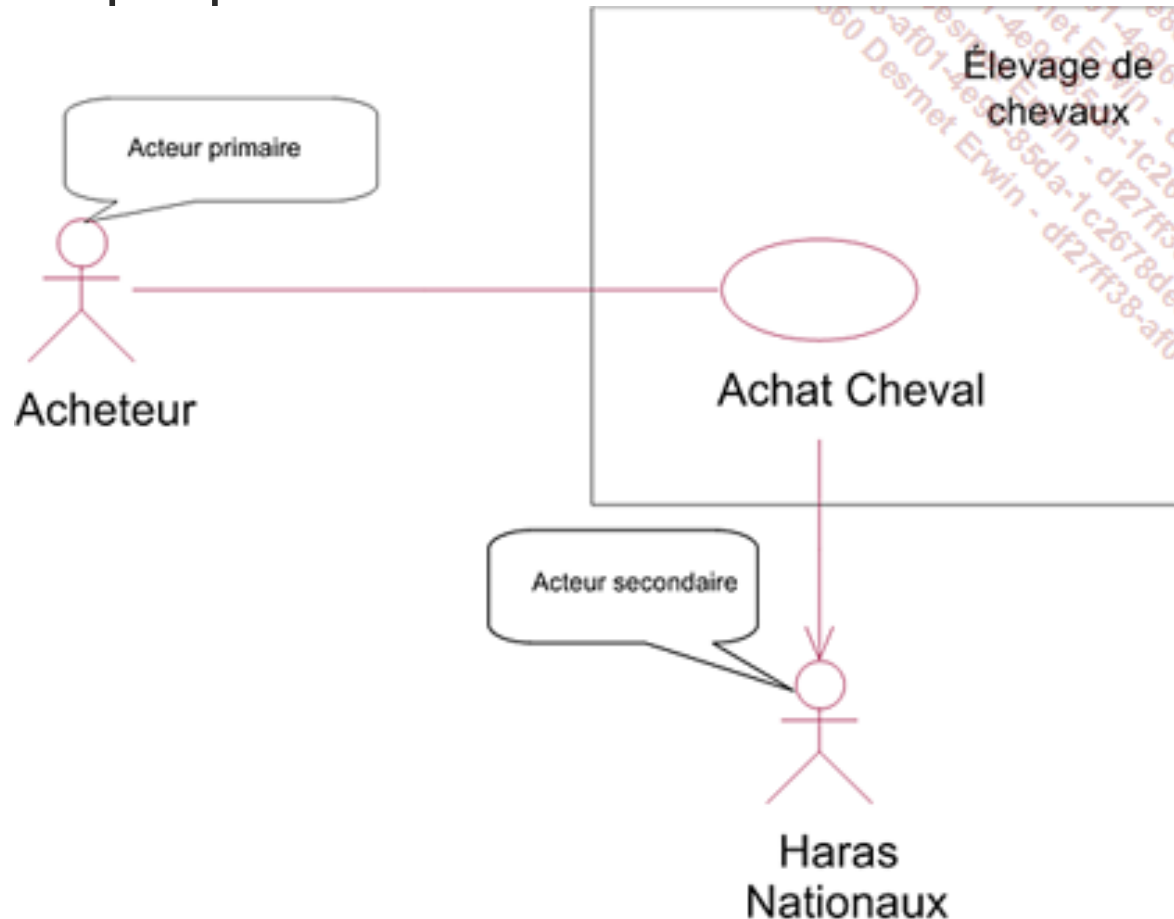


- Système qui répond au cas = Un rectangle (pas obligatoire)



## Modélisation des exigences

- Acteur secondaire même représentation
- Différence est que pour un acteur secondaire on a d'office un sens



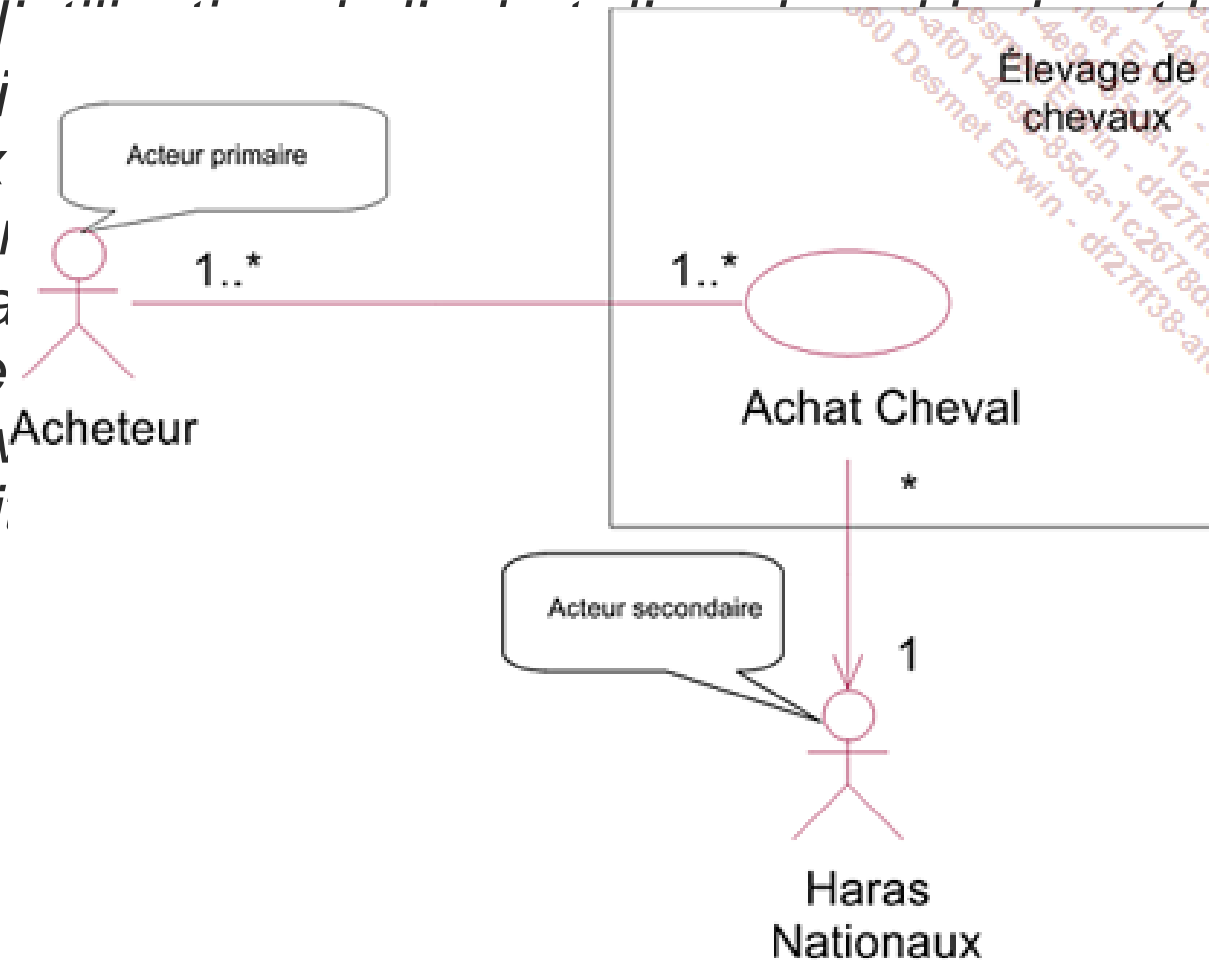
# Modélisation des exigences

Spécification	Cardinalités
0..1	zéro ou une fois
1	une et une seule fois
*	de zéro à plusieurs fois
1..*	de une à plusieurs fois
M..N	entre M et N fois
N	N fois



# Modélisation des exigences

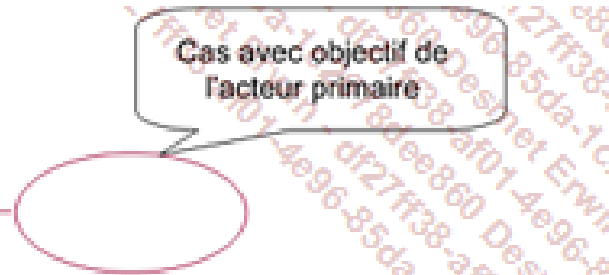
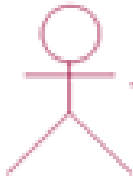
- Le cas d'achat de chevaux par les Haras Nationaux



# Modélisation des exigences : relations entre les cas d'utilisation

## • L'inclusion

- Enrichissement
- Elle
- On
- Part
- RepAcheteur
- « inc



Achat Étalon

<<include>>



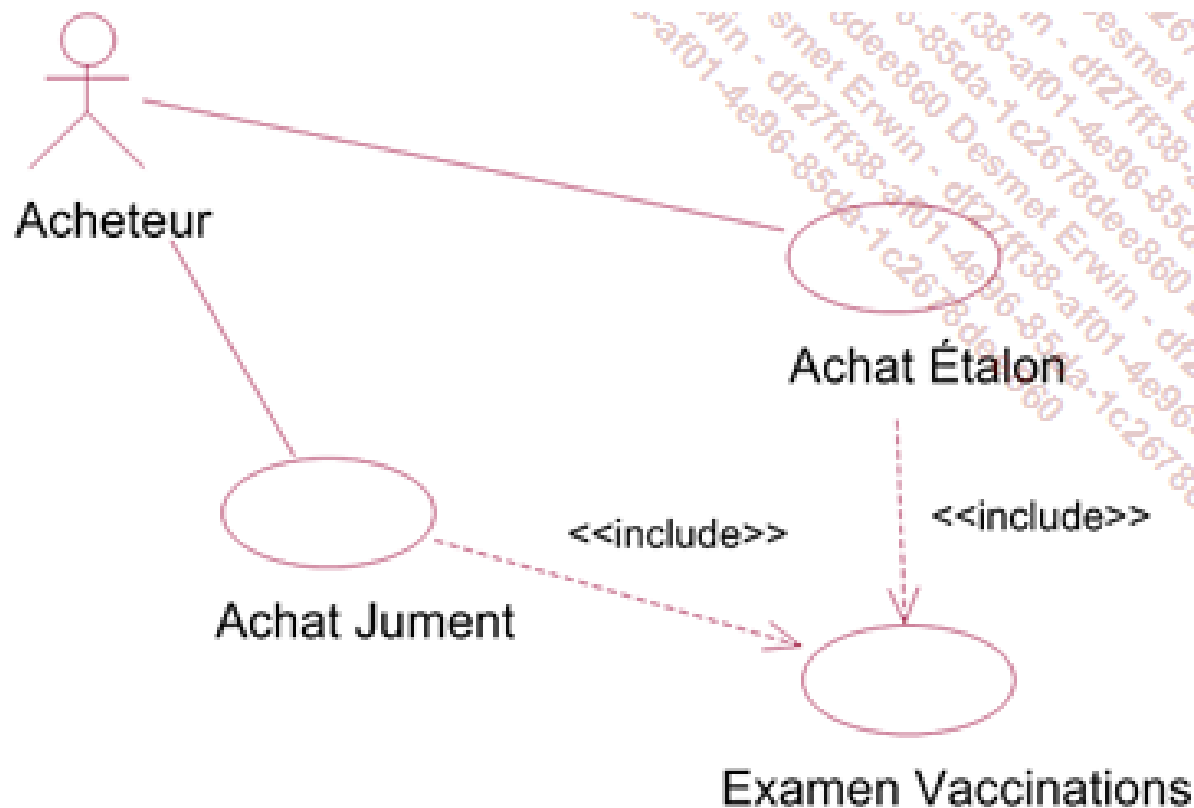
Sous-fonction  
d'utilisation incluse

Examen Vaccinations

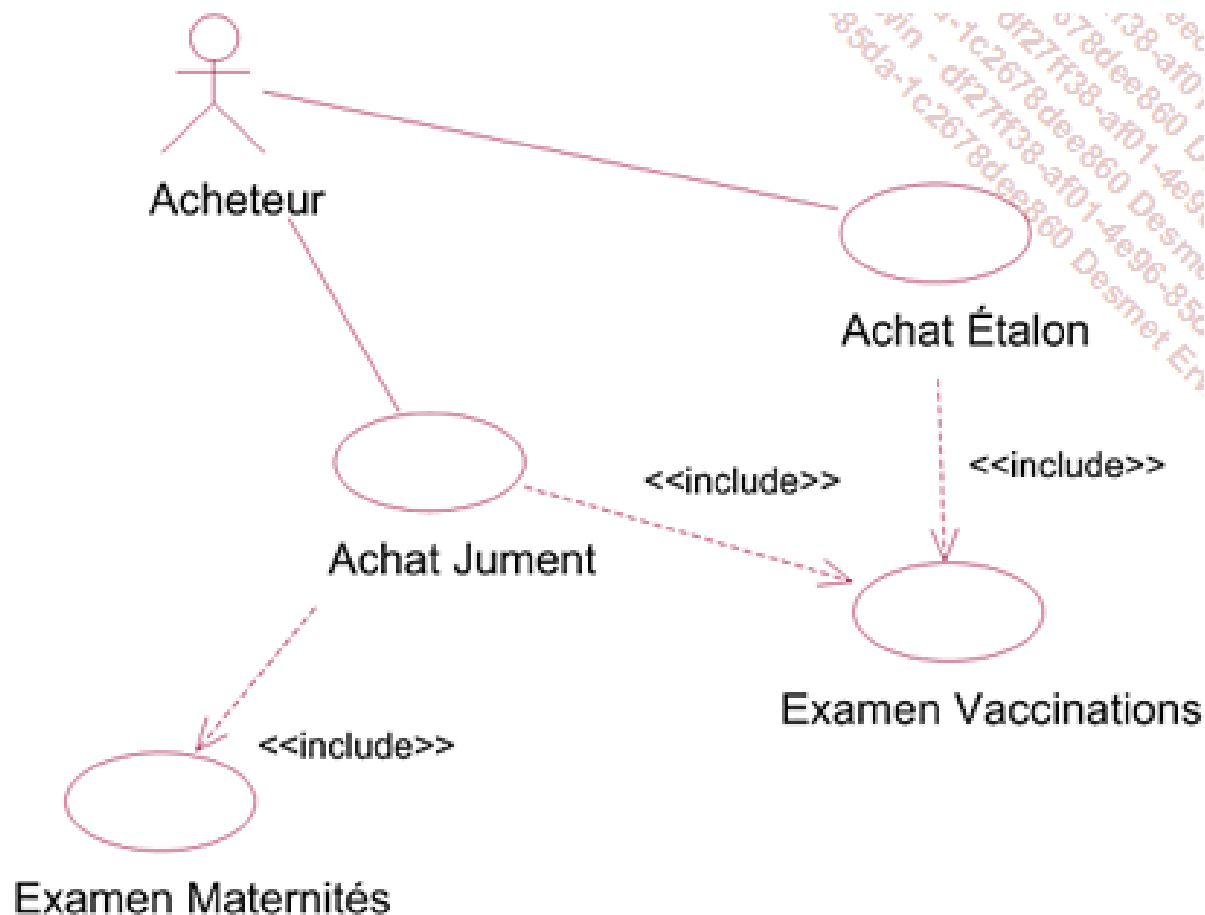
ie ses  
d'un

- Exel  
vacc  
étal

- Exemple avec mise en commun du sous cas

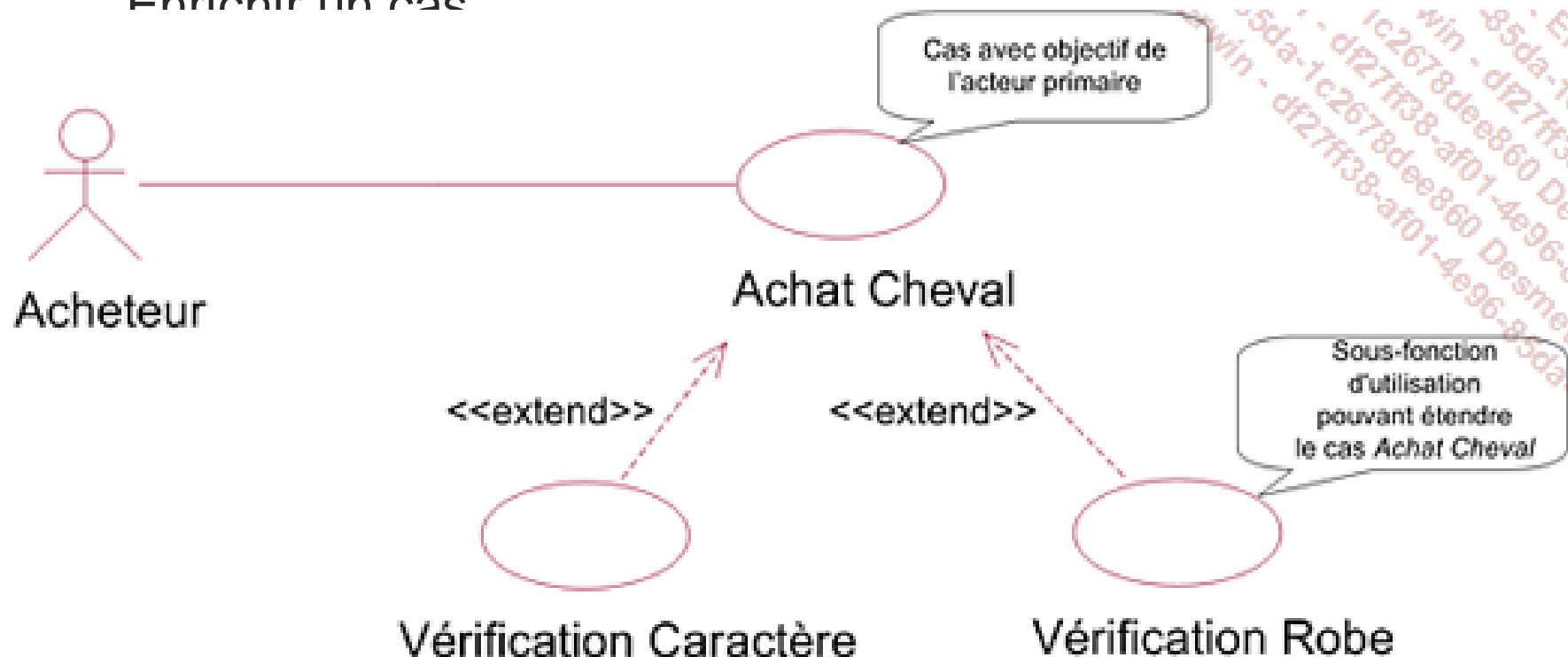


- Sous cas-non partagé, on parle de décomposition



# Modélisation des exigences : relations entre les cas d'utilisation

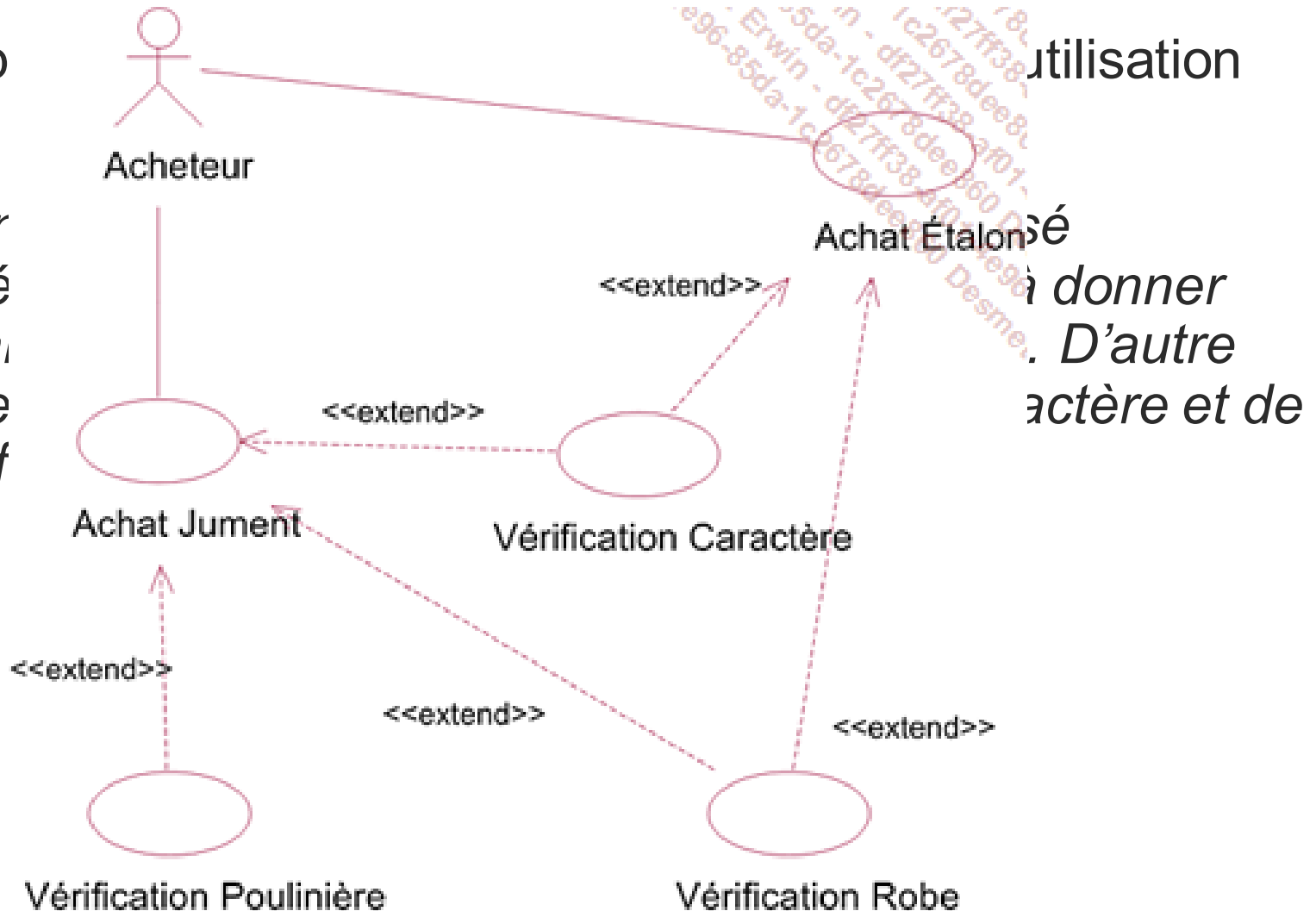
- L'extension
  - Enrichir un cas



# Modélisation des exigences : relations entre les cas d'utilisation

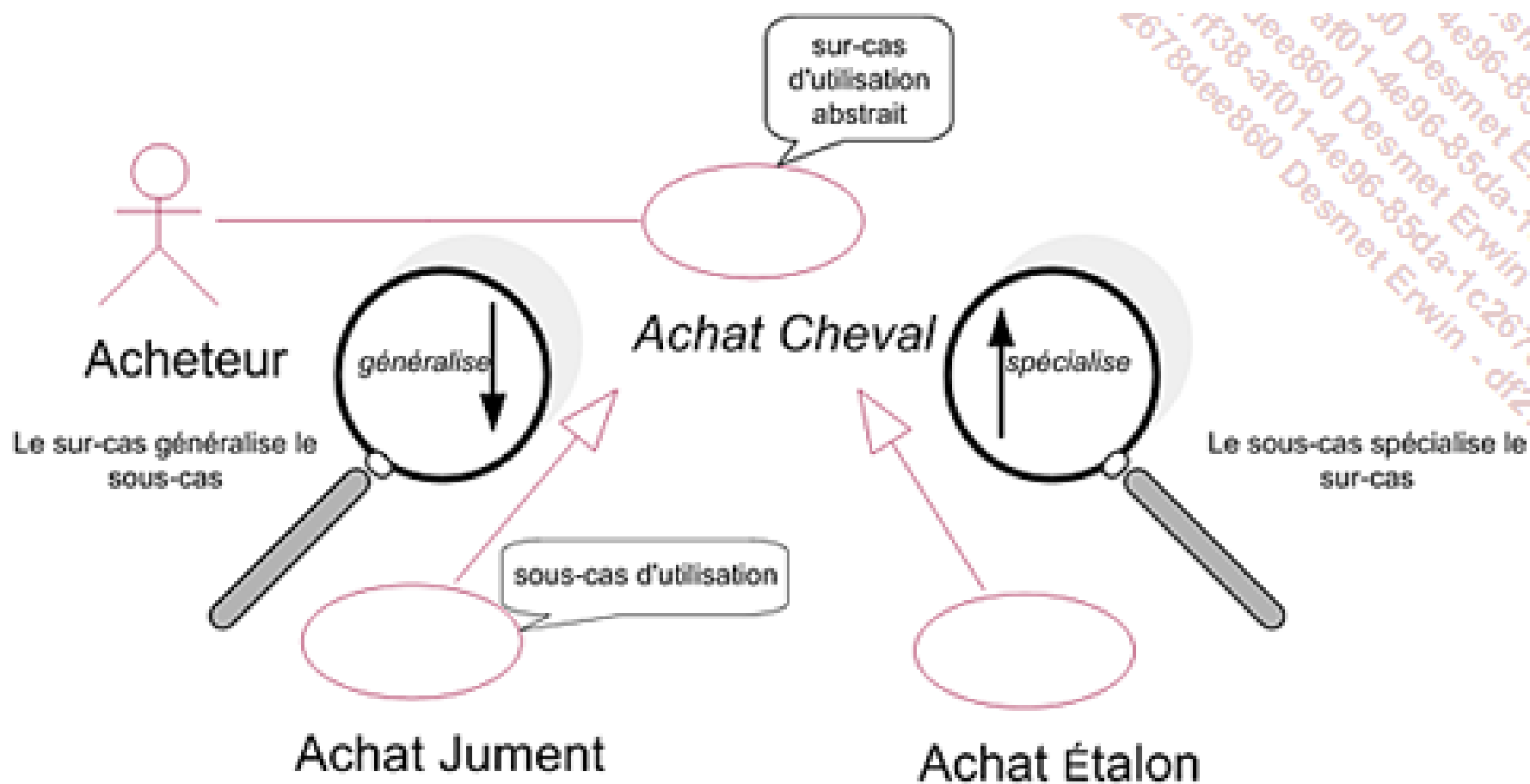
- Exemp

- Prenor  
séparé  
naissai  
part, le  
la vérif*



- Spécialisation d'un cas en un autre
- Le sous-cas hérite des comportements du sur-cas
- Egalement des associations et des relations d'inclusion/extension
- Sur-cas souvent abstrait (comportement partiel)
- Sous-cas à le niveau du sur-cas → si liaison avec acteur c'est pareil, si sous-fonction c'est pareil

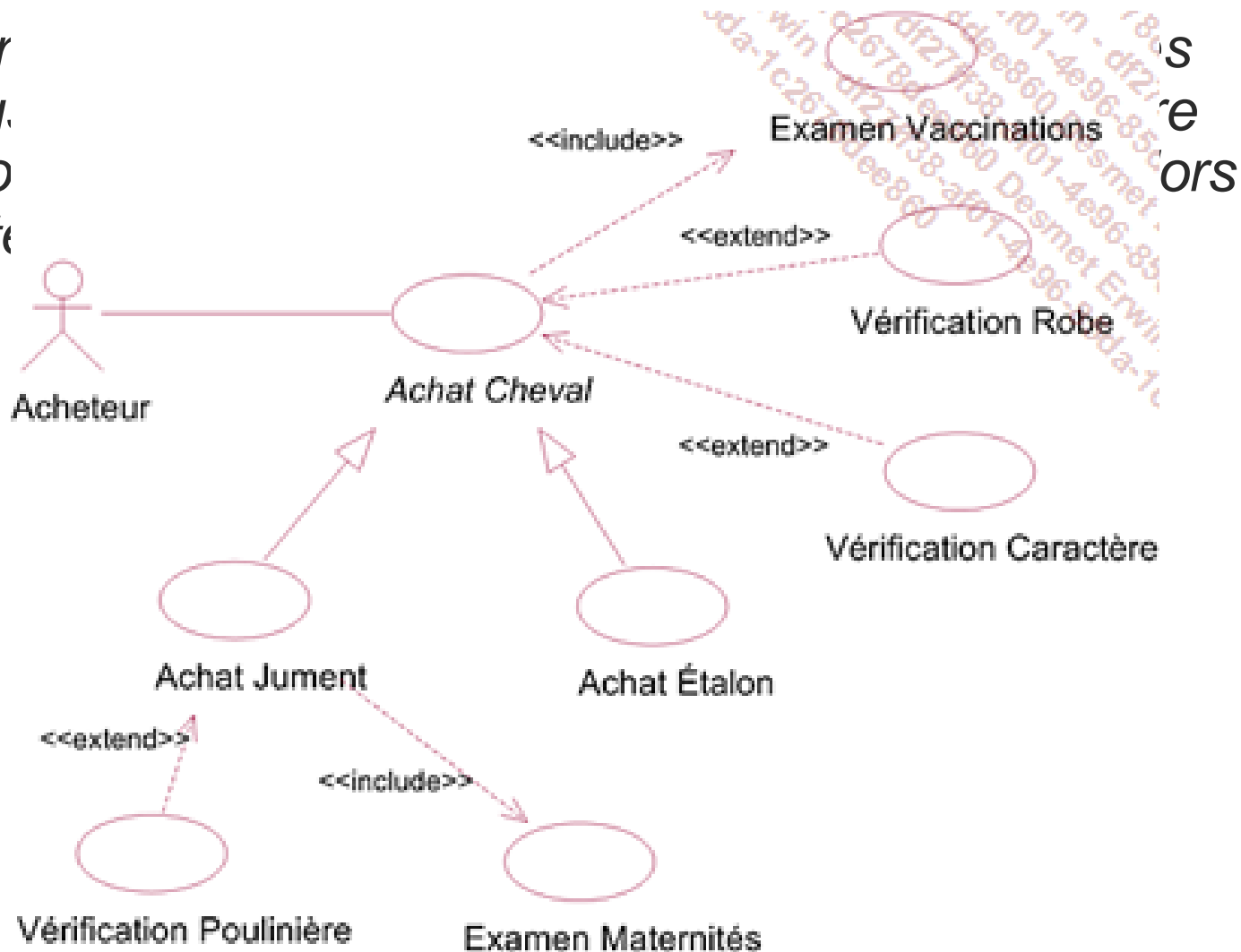
# Modélisation des exigences





# Modélisation des exigences

- Les relations inclure, étendre, hériter



- Ne fait pas partie de UML mais souvent utilisée
- Donner la description du comportements, des actions, des réactions
  - Le nom du cas d'utilisation.
  - L'acteur primaire.
  - Le système concerné par le cas d'utilisation.
  - Les intervenants (ensemble des acteurs).
  - Le niveau du cas d'utilisation pouvant être :
    - soit un objectif de l'acteur primaire.
    - soit une sous-fonction.
  - Les préconditions qui sont les conditions à remplir pour que le cas d'utilisation puisse être exécuté.
  - Les opérations du scénario principal.
  - Les extensions.

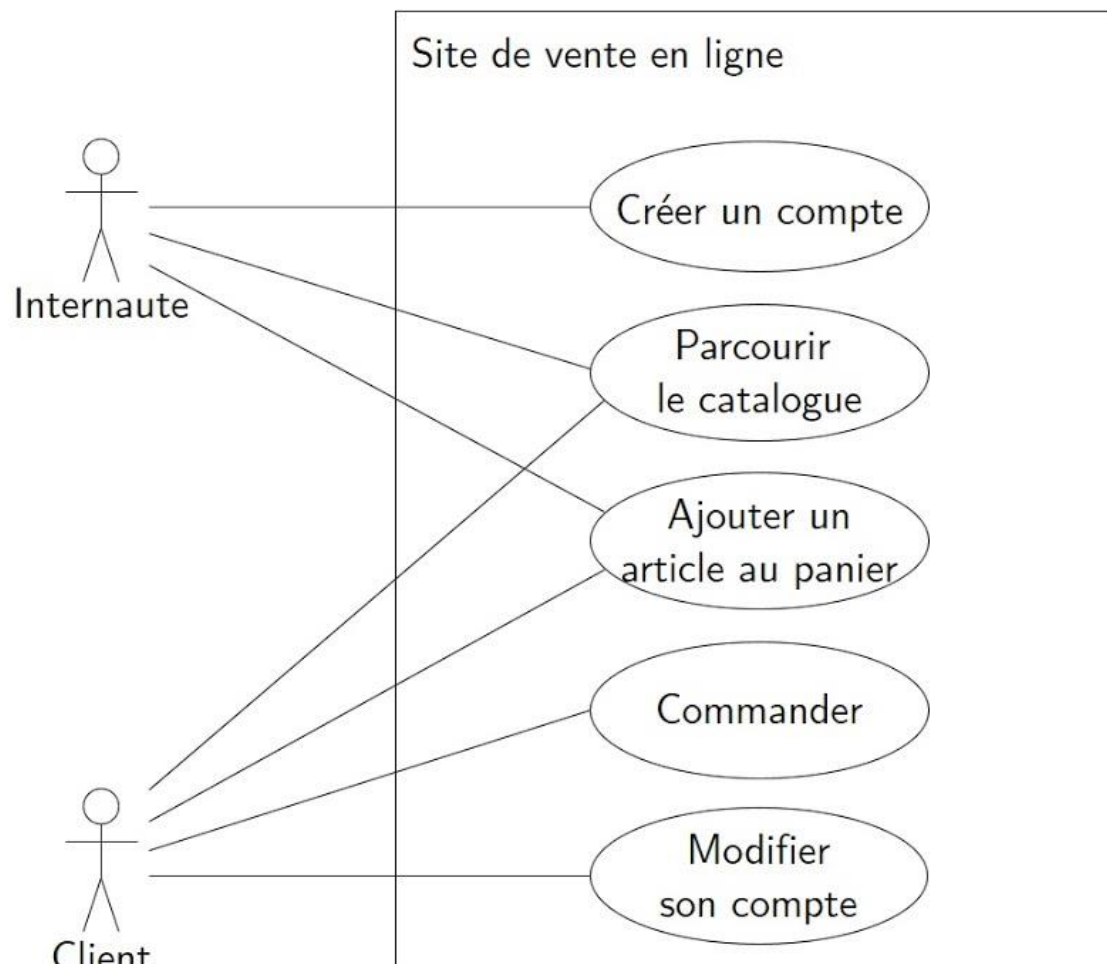
Cas d'utilisation	Nom du cas d'utilisation
Acteur primaire	Nom de l'acteur primaire
Système	Nom du système
Intervenants	Nom des intervenants
Niveau	Objectif de l'acteur primaire ou sous-fonction
Préconditions	Conditions devant être remplies pour exécuter le cas d'utilisation
Opérations	
1	Opération 1
2	Opération 2
3	Opération 3
4	Opération 4
5	Opération 5
Extensions	
1.A	Condition d'application de l'extension A sur l'opération 1
1.A.1	Opération 1 de l'extension A sur l'opération 1
1.A.2	Opération 2 de l'extension A sur l'opération 1
1.B	Condition d'application de l'extension B sur l'opération 1
1.B.1	Opération 1 de l'extension B sur l'opération 1
4.A	Condition d'application de l'extension A sur l'opération 4
4.A.1	Opération 1 de l'extension A sur l'opération 4

- Exemple à mettre en place :
- *Le cas d'utilisation d'achat d'une jument est illustré ci-après. Chaque extension porte le numéro de la ligne de l'opération à laquelle elle s'applique suivie d'une lettre qui permet de la distinguer d'une autre extension portant sur la même ligne. Ensuite, chaque opération d'une extension est numérotée, de la même façon que les opérations du scénario principal.*

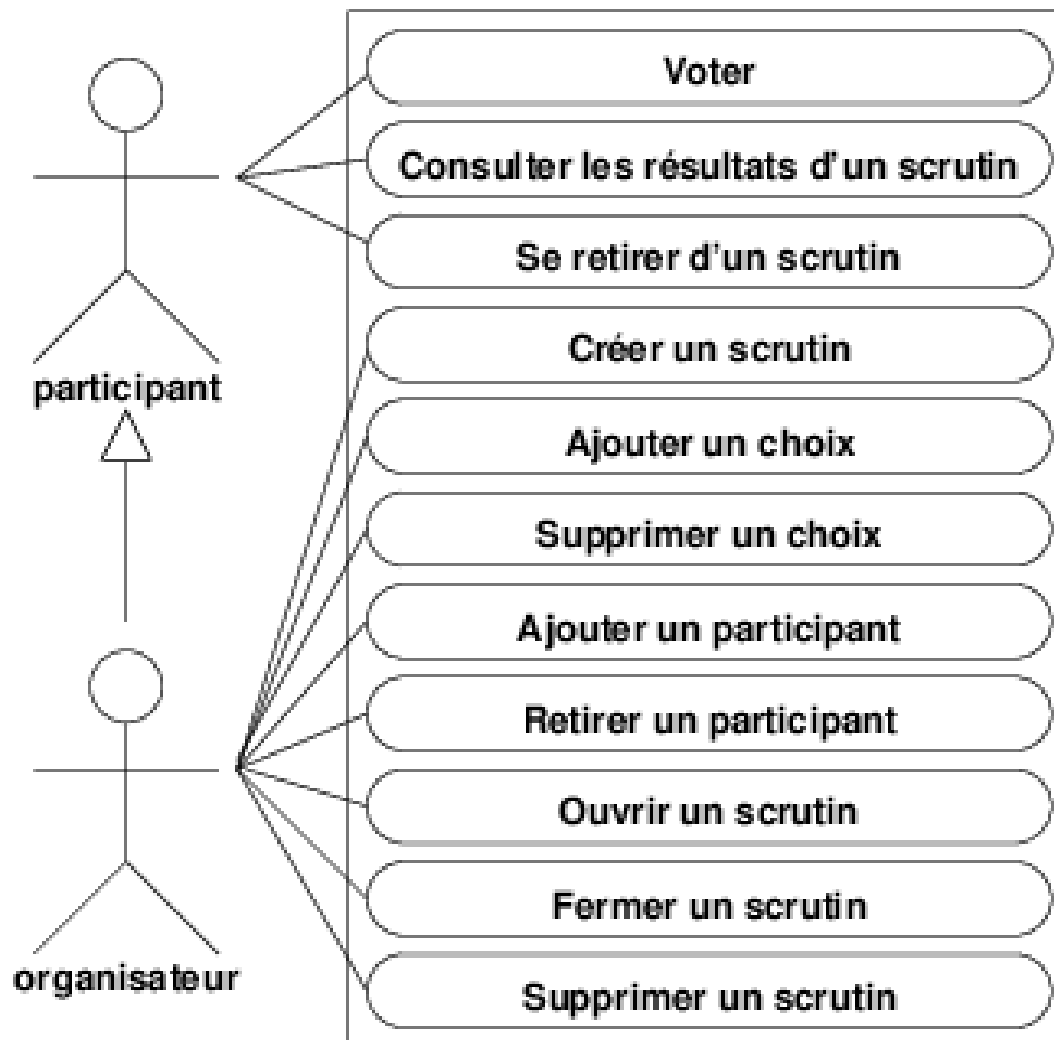
Cas d'utilisation	Achat d'une jument
Acteur primaire	Acheteur
Système	Élevage de chevaux
Intervenants	Acheteur, Haras nationaux
Niveau	Objectif de l'acteur primaire
Précondition	La jument est à vendre
Opérations	
1	Choisir la jument
2	Vérifier les vaccinations
3	Examiner les maternités
4	Recevoir une proposition de prix
5	Évaluer la proposition de prix
6	Payer le prix de la jument
7	Remplir les papiers de vente
8	Enregistrer la vente auprès des haras nationaux
9	Aller chercher la jument
10	Transporter la jument
Extensions	
2.A	Les vaccinations conviennent-elles ?
2.A.1	Si oui, continuer
2.A.2	Si non, abandonner
3.A	L'examen des maternités convient-il ?
3.A.1	Si oui, continuer
3.A.2	Si non, abandonner

- Les cas d'utilisation servent à :
  - Exprimer les exigences fonctionnelles conférées au système par les utilisateurs lors de la rédaction du cahier des charges.
  - Vérifier que le système répond à ces exigences lors de la livraison.
  - Déterminer les frontières du système.
  - Écrire la documentation du système.
  - Construire les jeux de test.

- [https://www.youtube.com/watch?v=GC5BdRve38A&ab\\_channel=DelphineLonguet](https://www.youtube.com/watch?v=GC5BdRve38A&ab_channel=DelphineLonguet)









- Use case c'est sympa même avec le tableau additionnel mais ça ne suffit pas pour développer.
- Suite de la vidéo d'aide :  
[https://www.youtube.com/watch?v=1G0omjzh1OQ&ab\\_channel=DelphineLonguet](https://www.youtube.com/watch?v=1G0omjzh1OQ&ab_channel=DelphineLonguet)

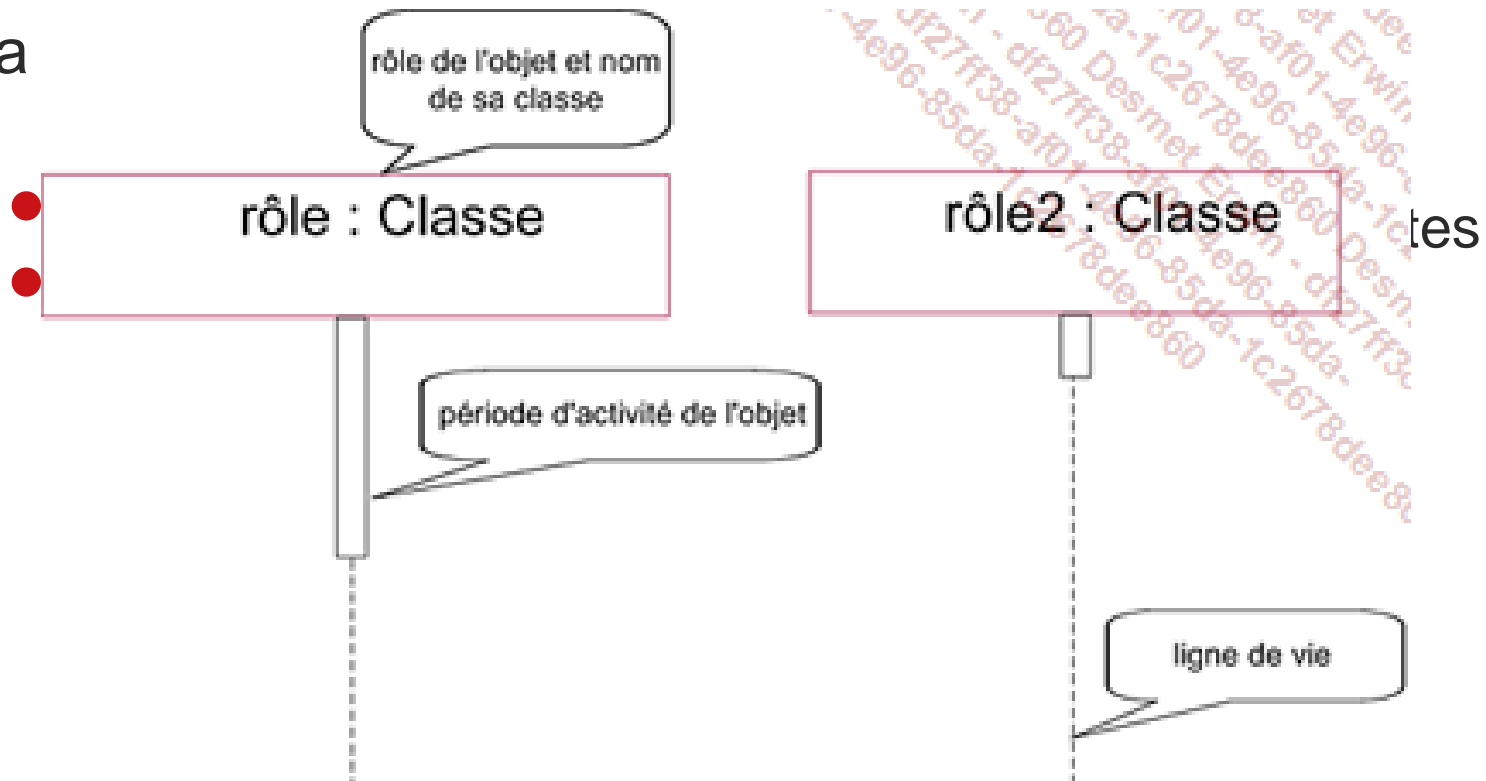
- Comment représenter les interactions entre les objets ?
- Un objet possède son propre comportement
- Comment représenter les interactions entre les objets ?
  - Diagramme de séquences (aspects temporelles)
  - Diagramme de communication (aspects spatiales)

## Diagramme de séquence

- Souvent sur plusieurs diagramme sauf pour petits systèmes
- Lie souvent aux sous-fonctions / Représente la dynamique
- Décrire les interactions entre un groupe d'objet
- Peut aussi montrer la transmission de données
- ***Pour interagir entre eux, les objets s'envoient des messages. Lors de la réception d'un message, un objet devient actif et exécute la méthode de même nom. Un envoi de message est donc un appel de méthode.***

# Diagramme de séquence : la ligne de vie

• La

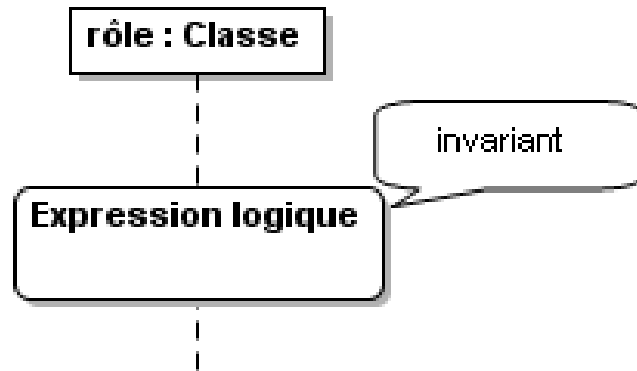


La notation **rôle : Classe** représente le rôle d'une instance suivi du nom de sa classe.

Par souci de simplification, nous considérons que le rôle de l'instance correspond à son nom. Le rôle de l'instance est optionnel si une seule instance de cette classe participe au diagramme de séquence.

Le nom de la classe peut également être omis dans le cas d'une étape préliminaire de la modélisation, mais il doit être spécifié dès que possible.

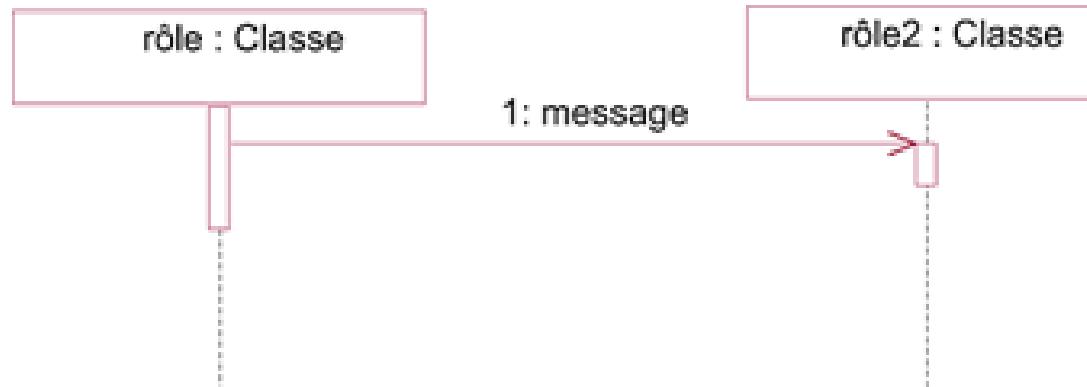
- Un diagramme de séquence contient plusieurs lignes de vie
- Elle peut par exemple commencer par un invariant d'état
  - Expression vérifier le long de la ligne de vie





## Diagramme de séquence : l'envoi de message

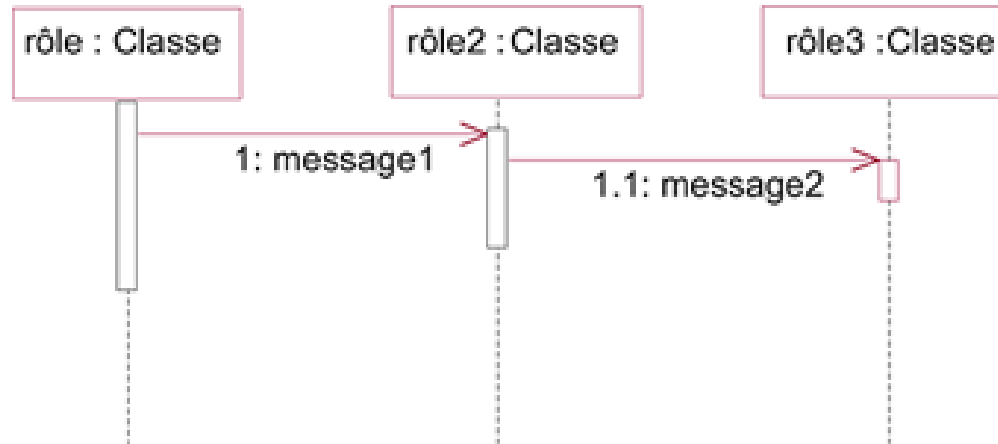
- Représenté par des flèches horizontales entre deux lignes de vies



- Objet de G envoie un message à l'objet de D
  - En prog on parlerait d'appel de la méthode **message**
  - Le nom du message est facultatif on peut mettre \*

## Diagramme de séquence : l'envoi de message

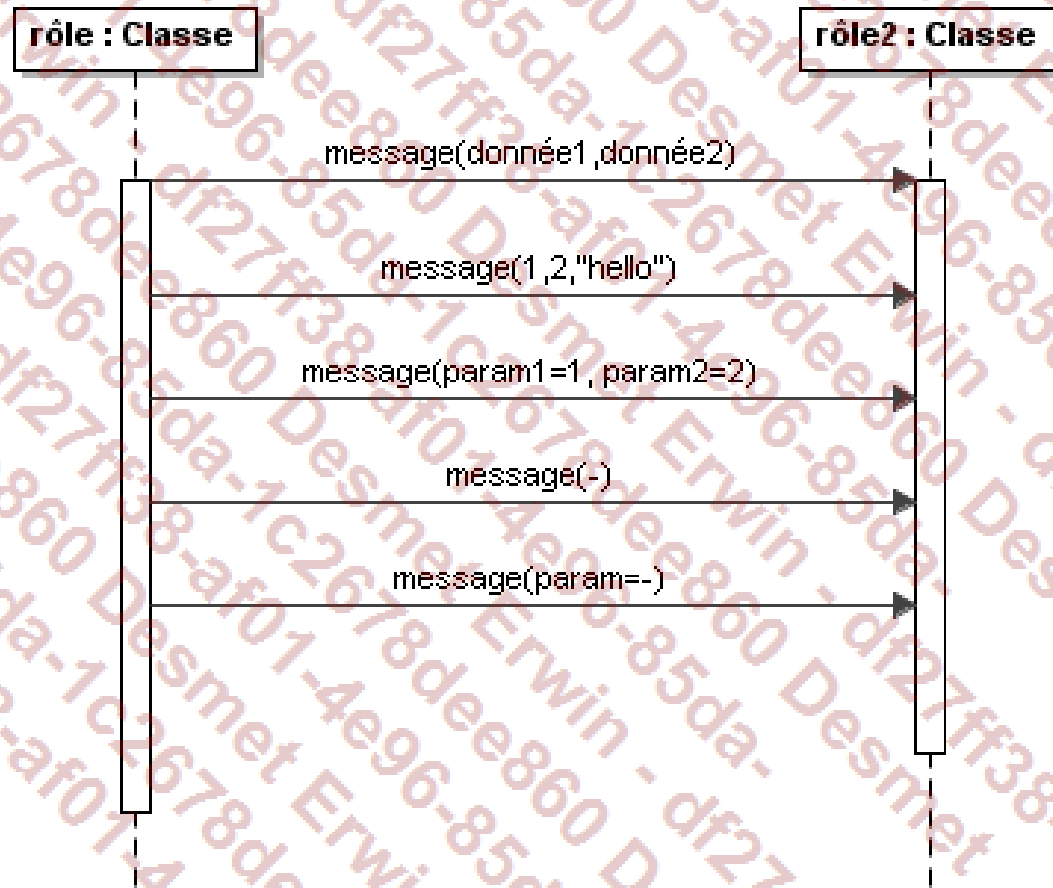
- Les messages sont numérotés à partir de 1
- Si pas séquentiel, on passe en numérotation composé



- La numérotation des messages n'est pas obligatoire. Elle reste toutefois pratique pour montrer les activations imbriquées.

# Diagramme de séquence : l'envoi de message

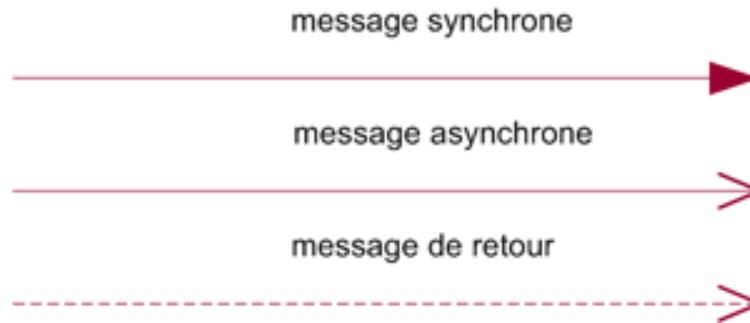
- On peut tr
- On le fait (
- Ajout du c  
fournie



ou cst)  
ir n'est pas

## Diagramme de séquence : l'envoi de message

- Différents types d'envois de messages



- Synchrone = le plus régulier, attente que l'activation de la méthode invoquée soit finie chez le destinataire
- Asynchrone = On attend pas la fin → Fct parallèle

# Diagramme de séquence : l'envoi de message

*Un cavalier donne un ordre à son cheval puis un second ordre sans attendre. Le message est asynchrone.*

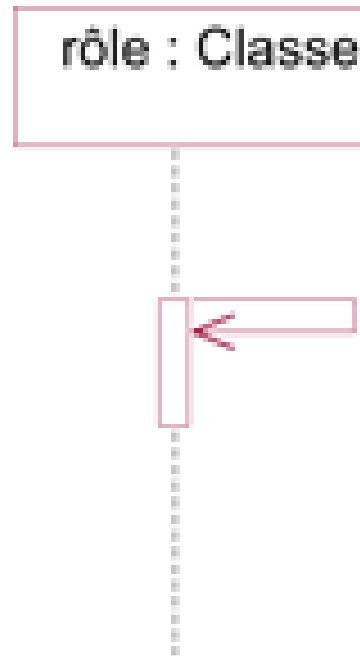


Le message  
pas système  
résultat.

le n'est  
nt pas un

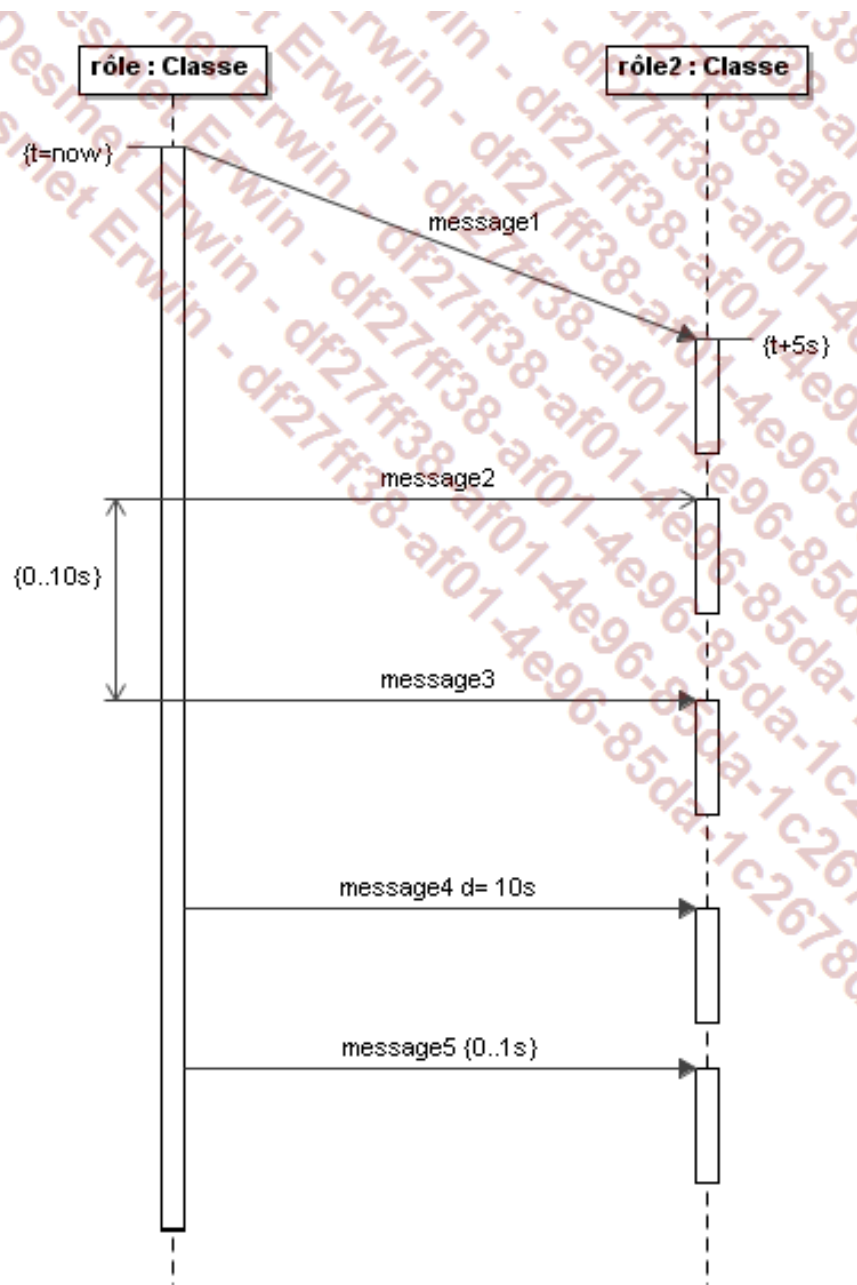
## Diagramme de séquence : l'envoi de message

- Un objet peut s'auto-envoyer un message



## • Introduction du t

- Le message 1 arrive 5 secondes plus tard
- Le message 3 arrive 10 secondes après le message 2 ;
- La transmission d'un paramètre de durée d'un message
- La durée de transmission d'un message est comprise entre 0 et 1 seconde.



ce : l'envoi

t) et il arrive 5

ndes après le

es. Le  
transmission

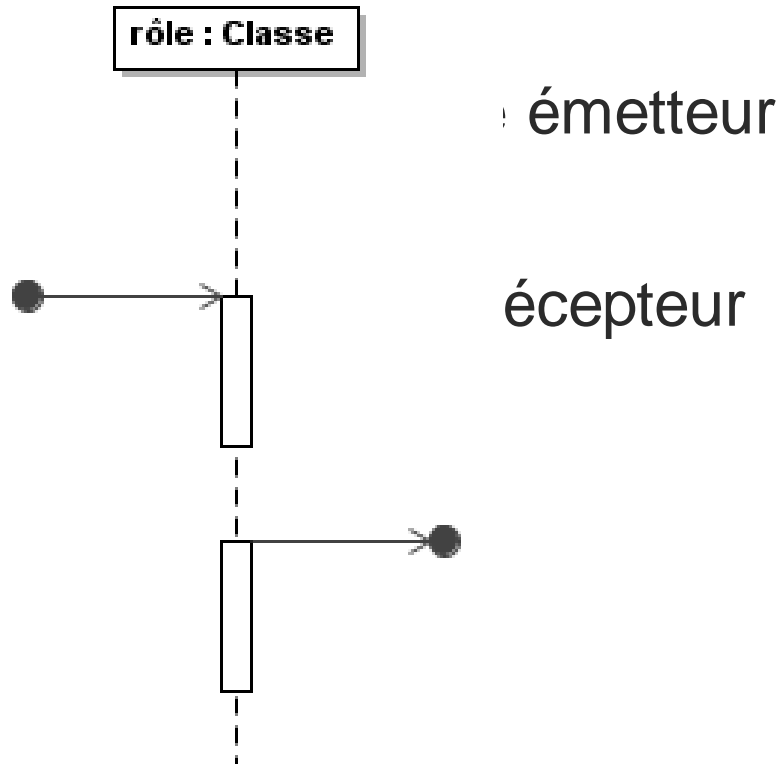
nprise entre 0

# Diagramme de séquence : l'envoi de message

- Message trouvé et message perdu

- Trouvé : Récepteur

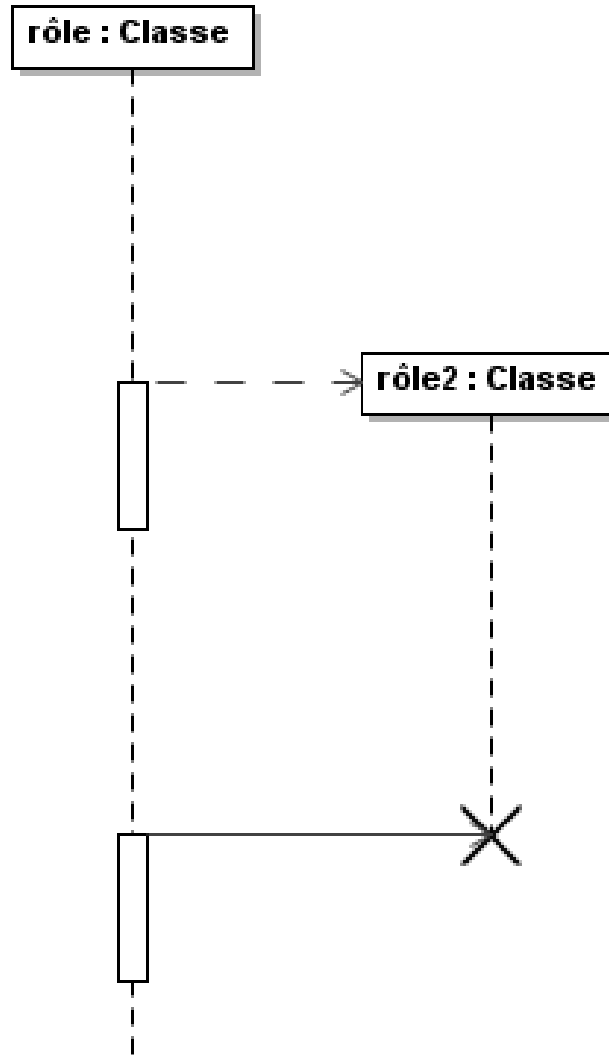
- Perdu : Emetteur c





# Diagramme de séquence : Création

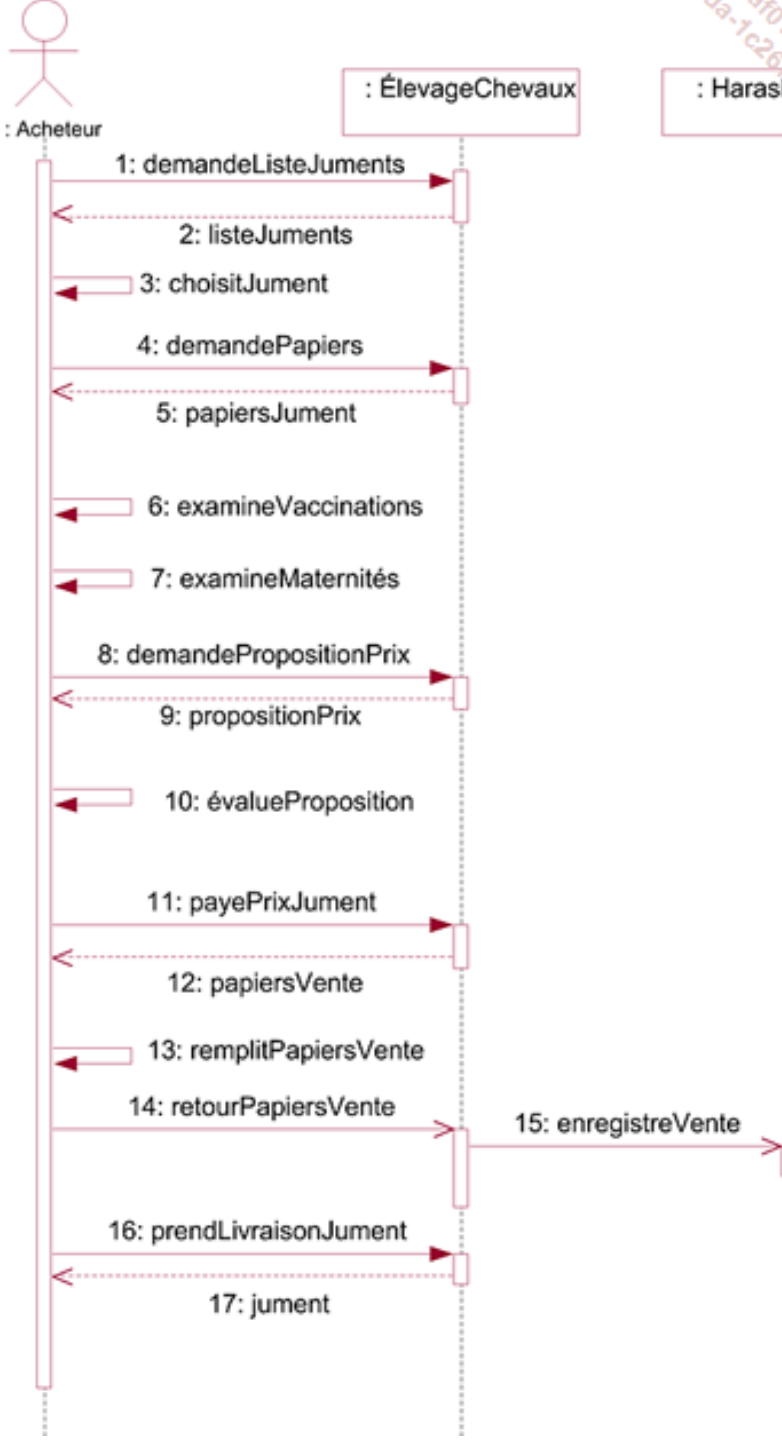
- Création = message vie
- Destruction = message



ou à la ligne de

n à la ligne de

- Exemple complètement étudié au chapitre 10, aucune alternative, nous verrons par la suite les boucles.



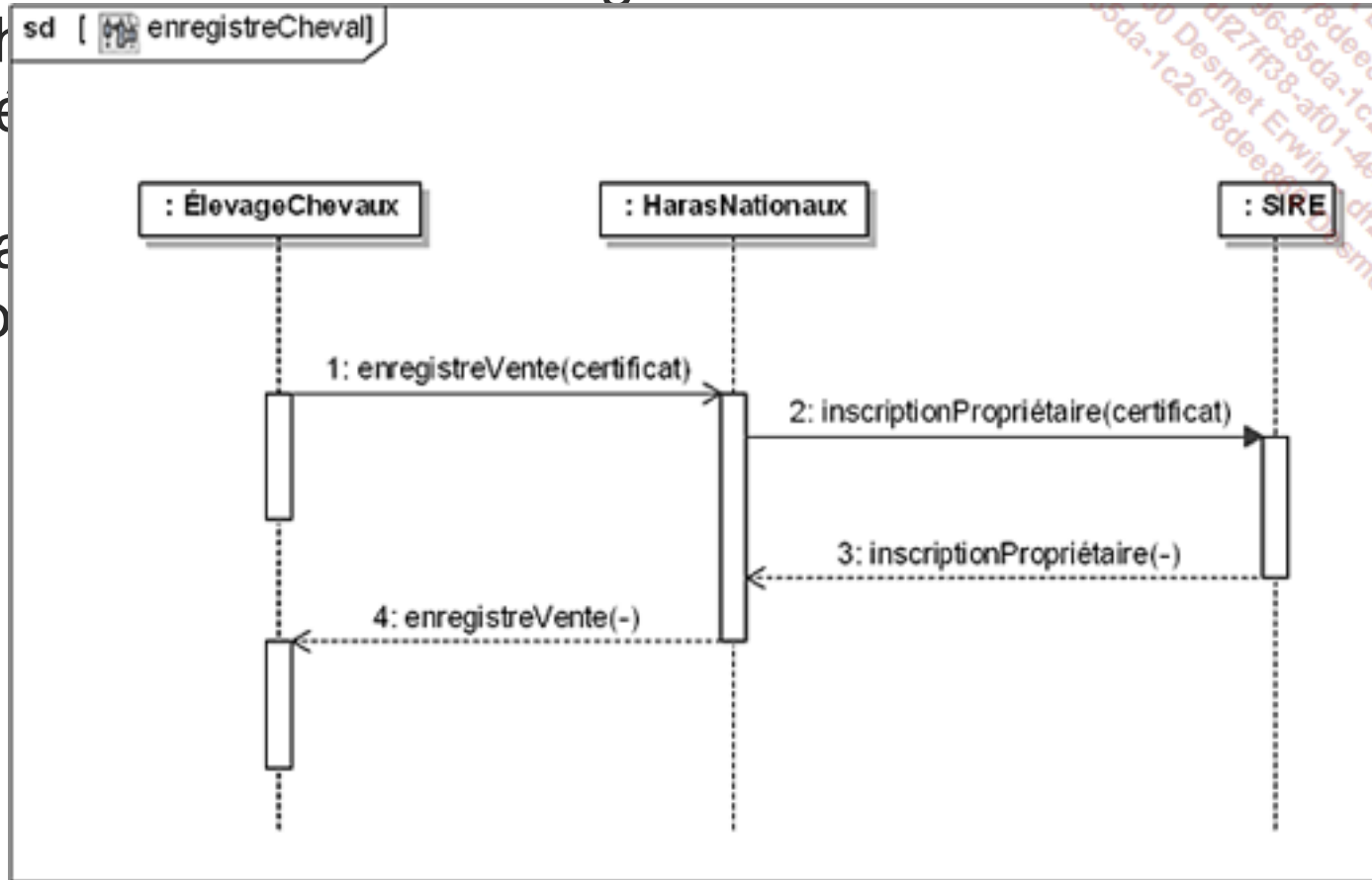
ce :  
nique

ent déjà  
ces. Il n'y a  
énario. Nous  
ernatives et

- Description de la dynamique d'un système
- La totalité est décrite par un ensemble de cadres
- On parle souvent de description modulaire
  - Référence à d'autres cadres
- Notation : un diagramme de séquences + un cartouche (nom + paramètres éventuels)

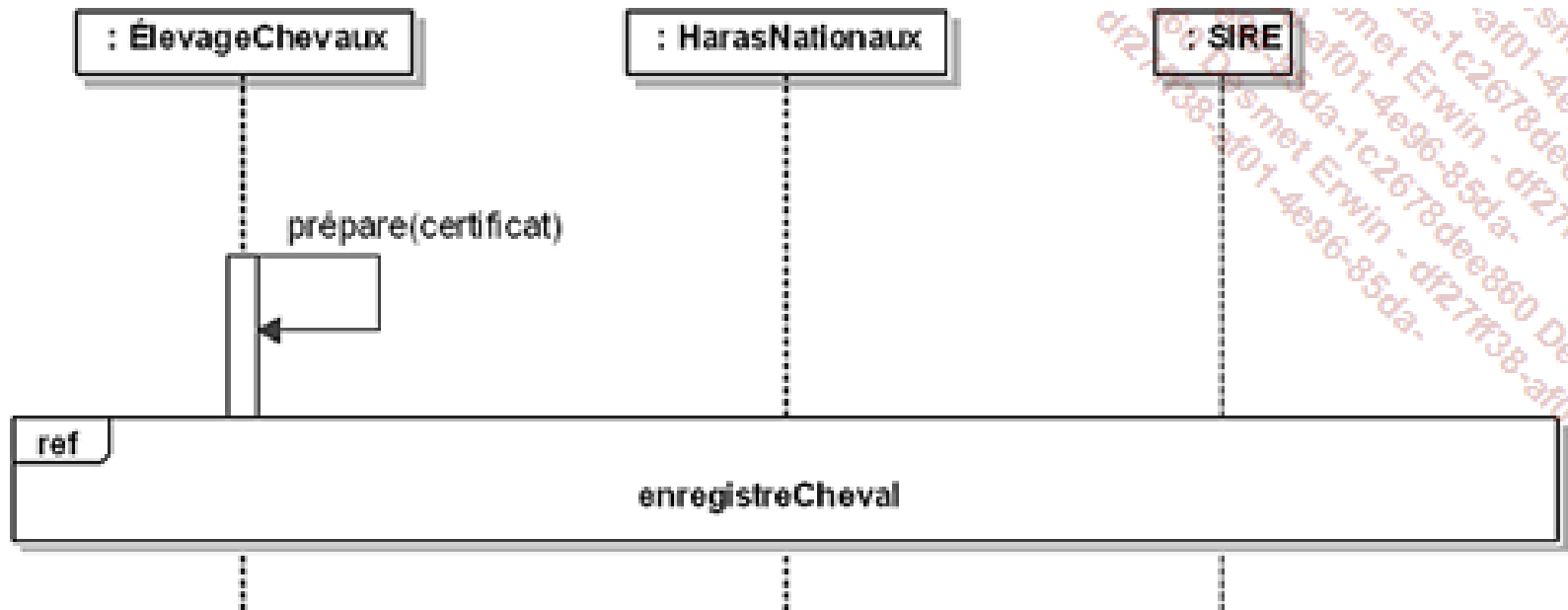
## Cadres d'interaction

Exemple de cadre d'interaction appelé `enregistreCheval` qui montre en détail l'enregistrement de la vente d'un cheval aux Haras Nationaux.

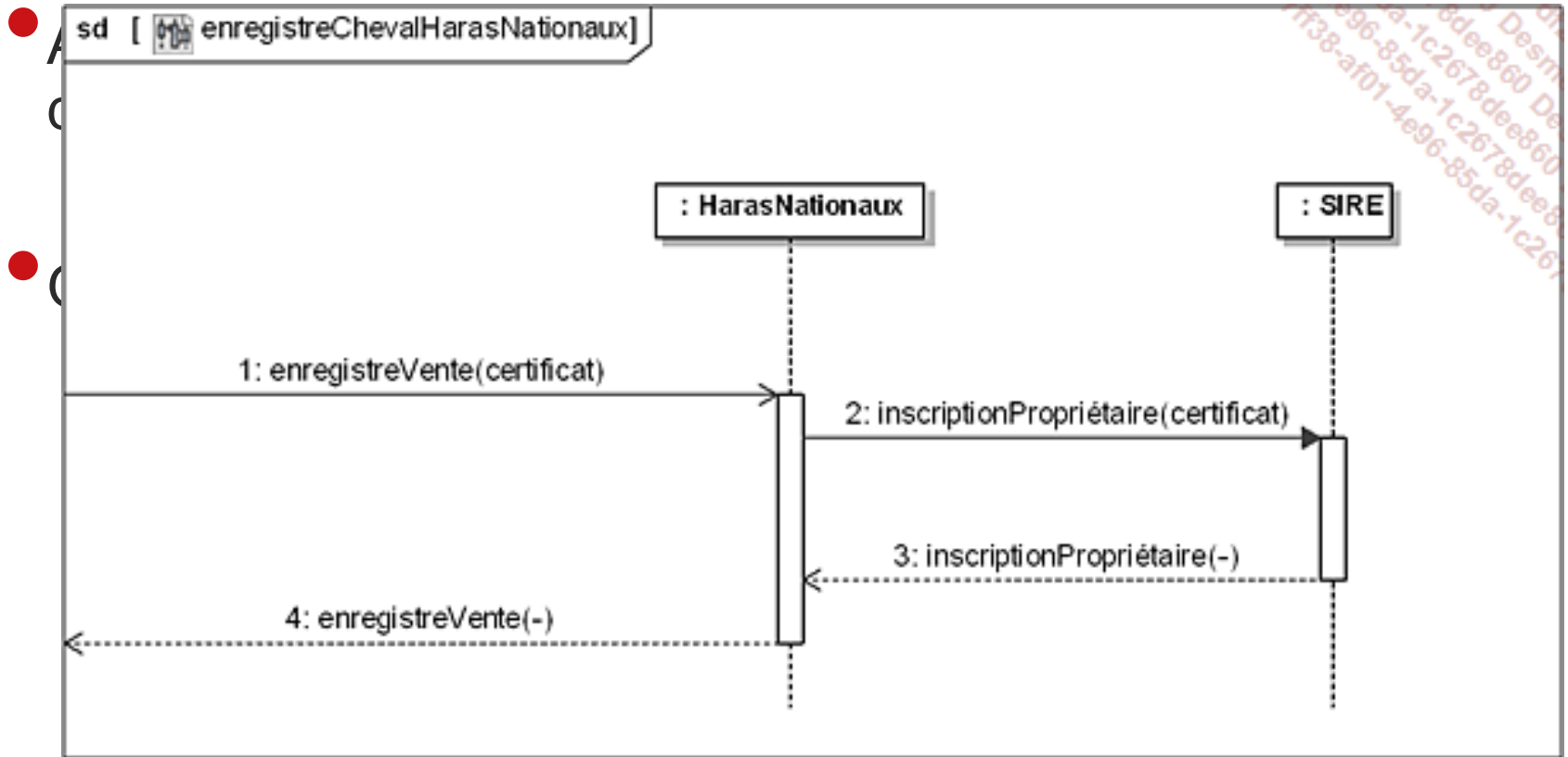


## Cadres d'interaction

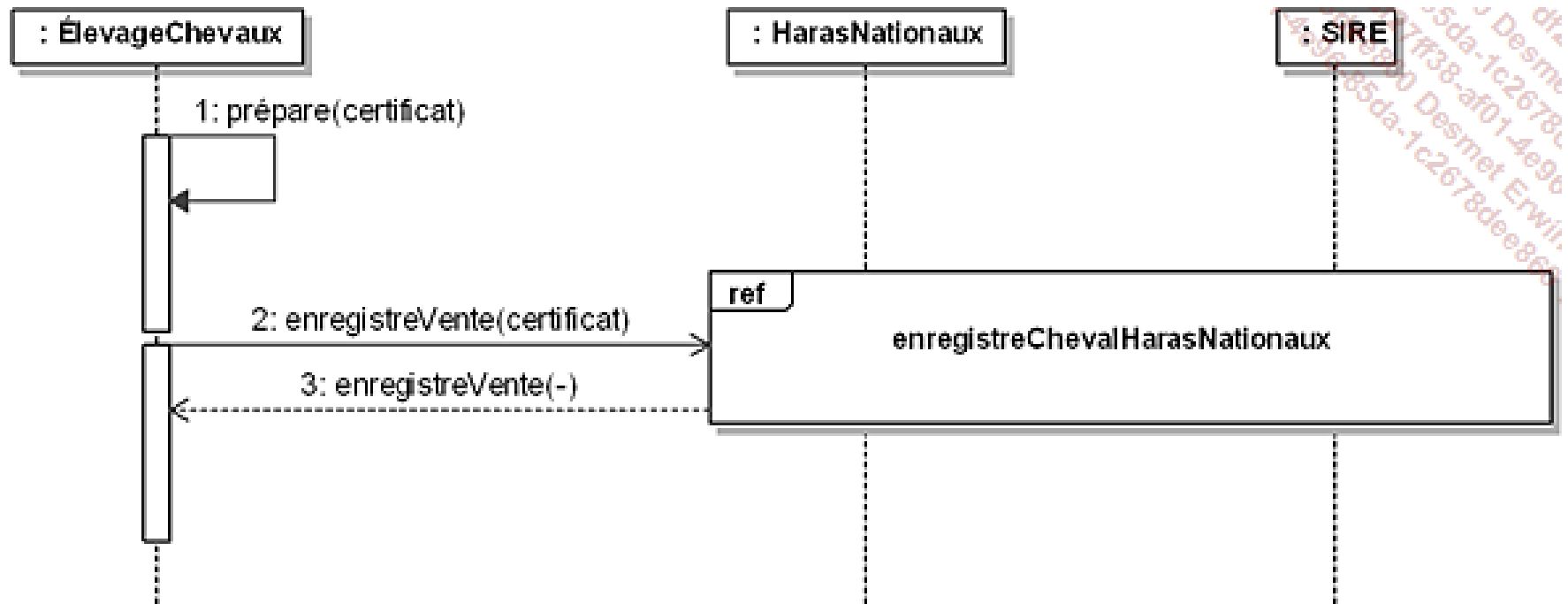
- Référence d'un cadre dans un autre avec simplification
- Lignes de vies doivent exister des deux cotés



# Cadres d'interaction



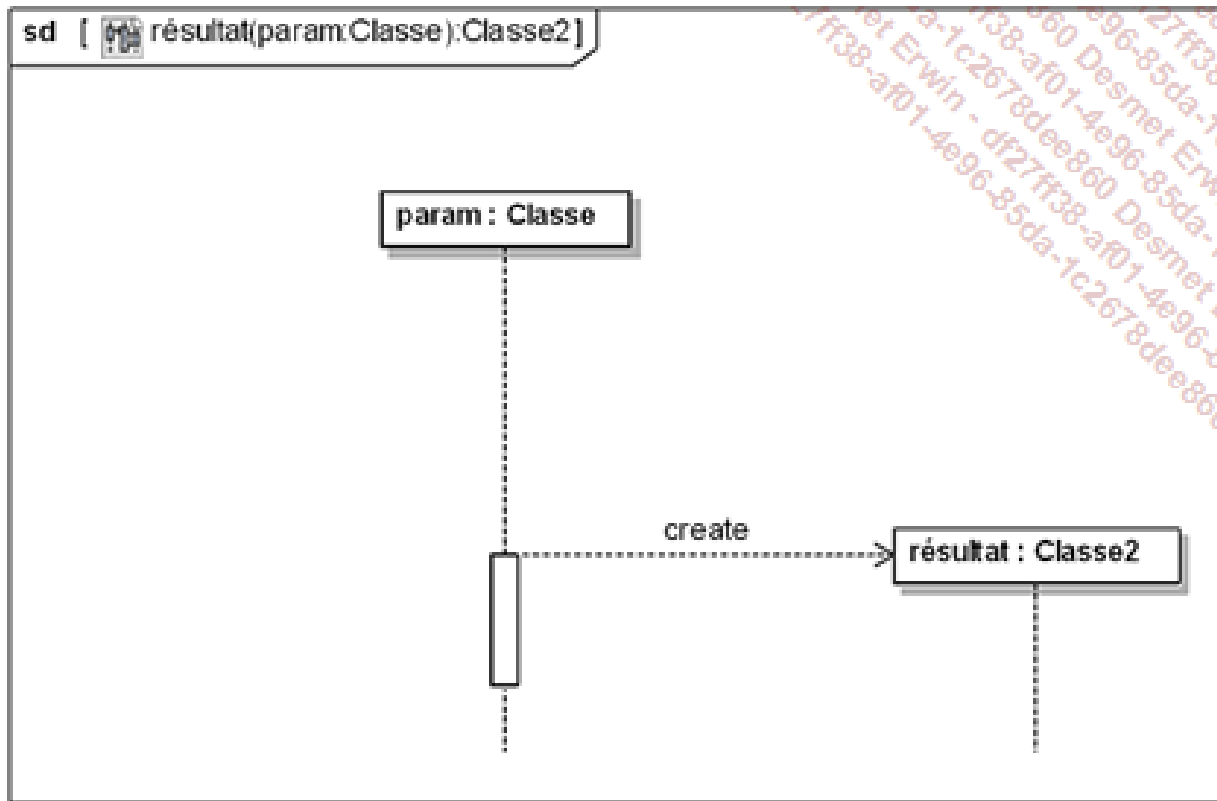
- Référençons l'exemple précédent



- Le passage de paramètre en entrée et le retour d'un résultat
- Nom du résultat= nom du cadre
- Paramètre entre parenthèse



- Param = instance de la classe Classe
- Résultat instance de la classe Classe2



- En programmation :
  - On utilise des expressions spécifiques

**[nomObjet.attribut opérateurAffectation] nomCadreInteraction  
[(listeParamètres)][ : valeurDeRetour]**

- nomObjet = nom d'un objet participant au diag de séquence
  - opérateurAffectation = «=», «+=», «-=», «\*=», «/=»
  - Valeur de retour pas obligatoire
  - Les parties entre[ ] sont facultatives
- Exemple → :Classe.résultat = additionne(1,2) : 3

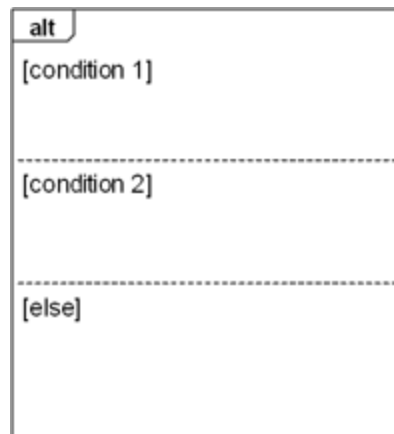
- Unités d'interaction
- Support aux boucles/parallélisme /alternatives
- Représente par un rectangle avec une étiquette
  - Contient un opérateur (détermine la modalité)
  - Souvent : option, boucle, alternative

## Les fragments combinés

- L'option : S'exécute si la condition est vraie



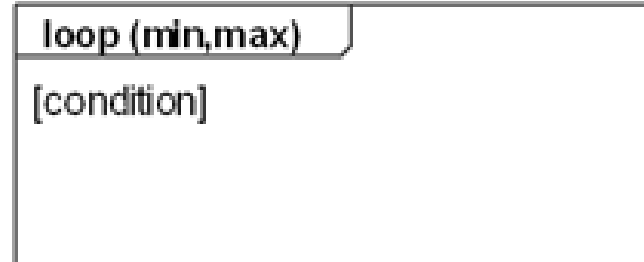
- L'alternative : Plusieurs conditions et un cas par défaut



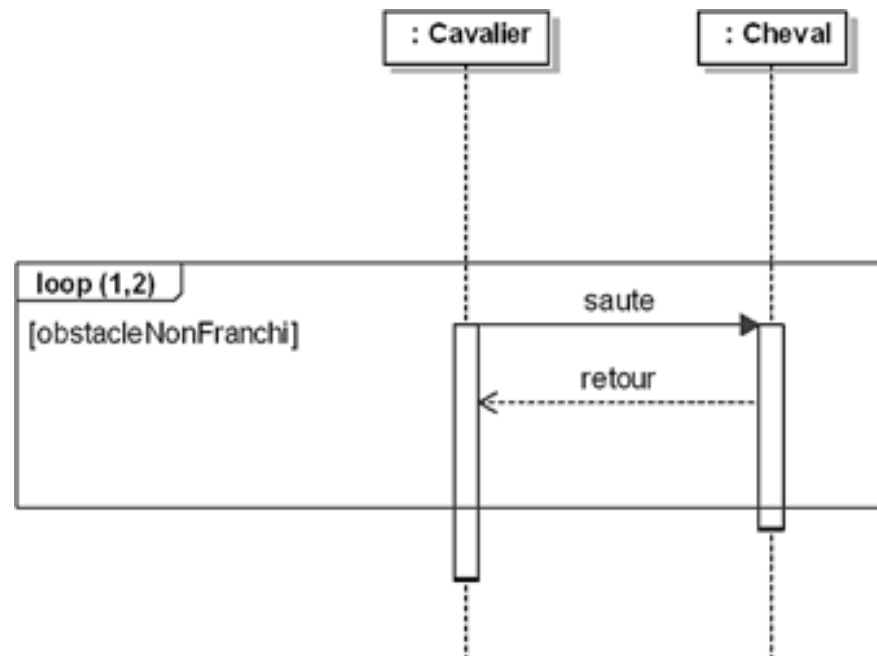
→ Else pas obligatoire

## Les fragments combinés

- La boucle : géré par l'opérateur **loop** et les param **min** et **max** associé à une condition

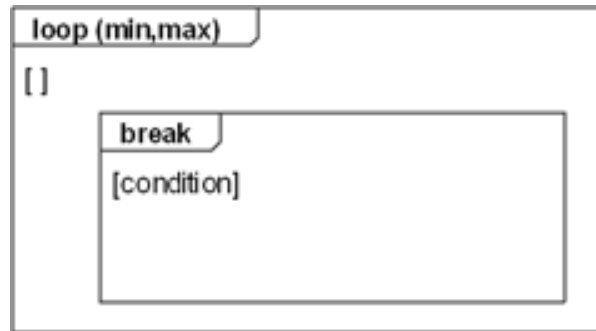


- Exemple : Pour franchir un obstacle, un cavalier peut s'y prendre à plusieurs reprises sans toutefois dépasser deux refus*



## Les fragments combinés

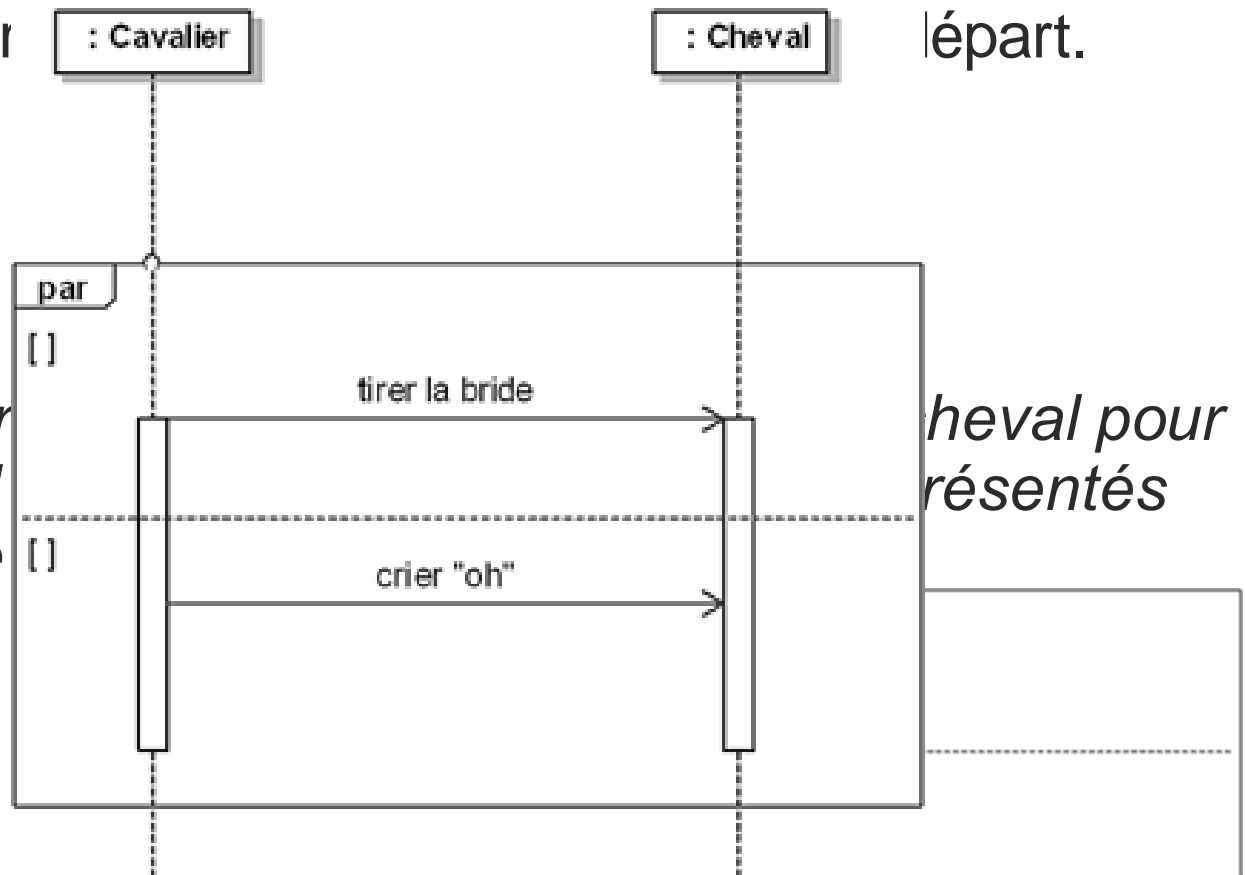
- L'opérateur break : ressemble à l'option mais ici après exécution, le fragment qui le contient se termine. Si aucun fragment, c'est le diagramme de séquence qui s'arrête



## Les fragments combinés

- Le parallélisme : fragment en plusieurs parties dont chaque contenu est exécuté simultanément.

Chaque par  
Départ pas

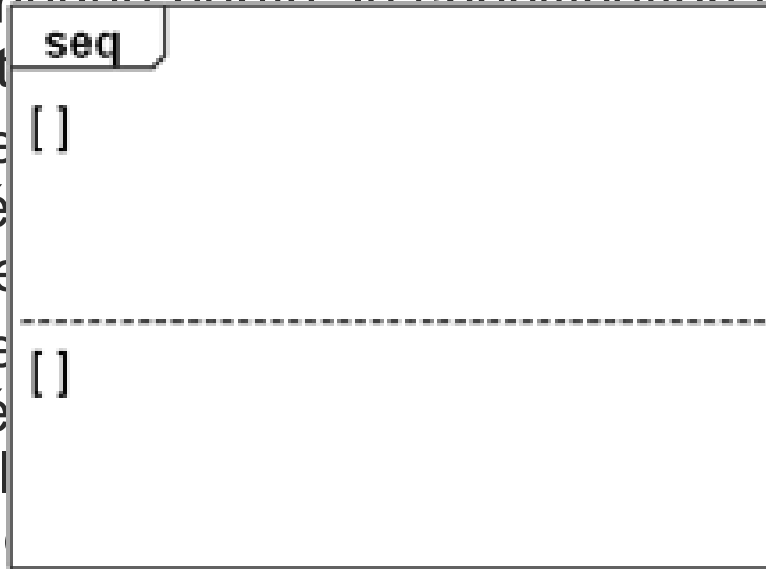


Exemple : les ordres  
l'arrêter alors qu'il  
par des envois de

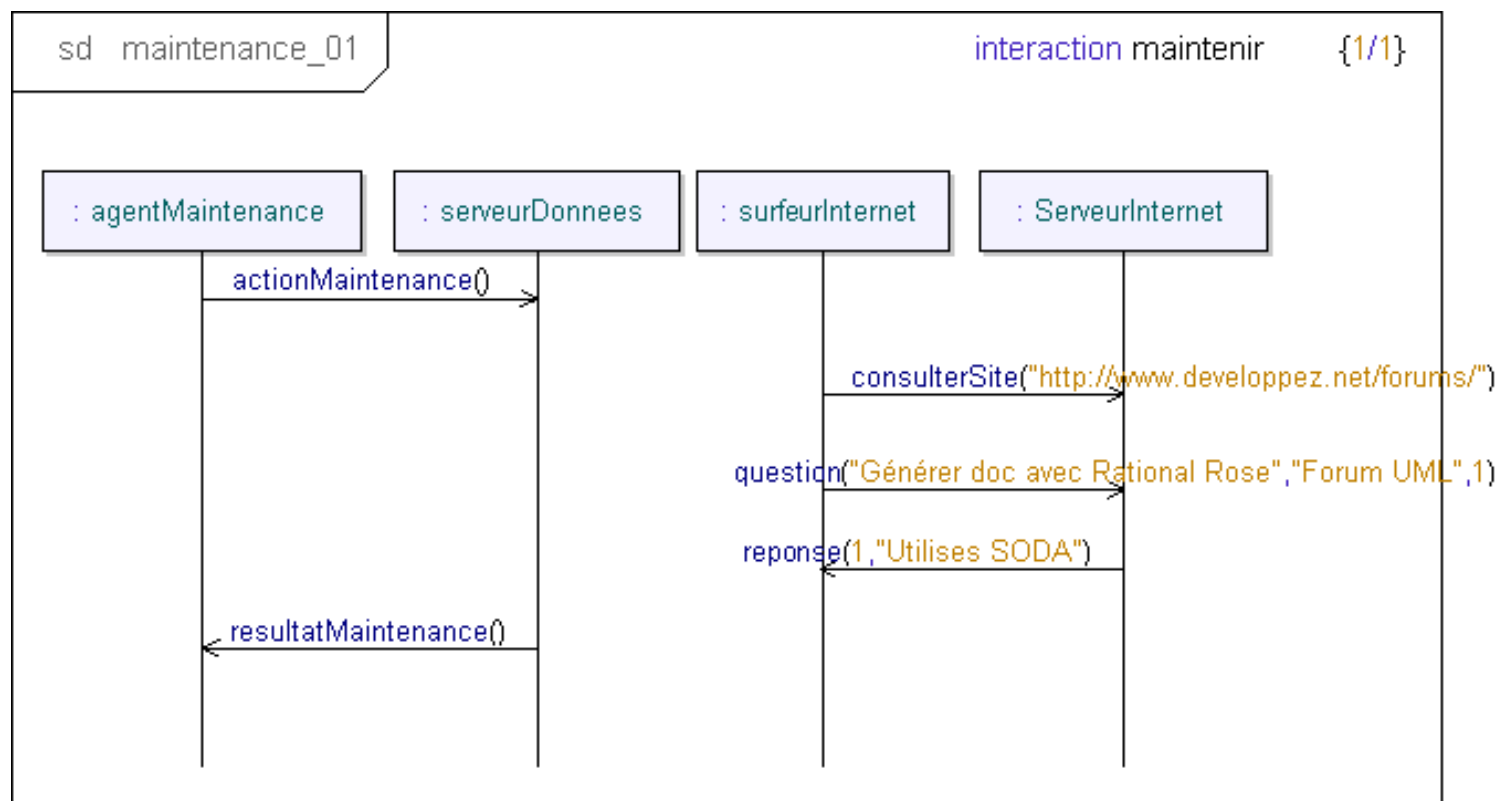
cheval pour  
présentés

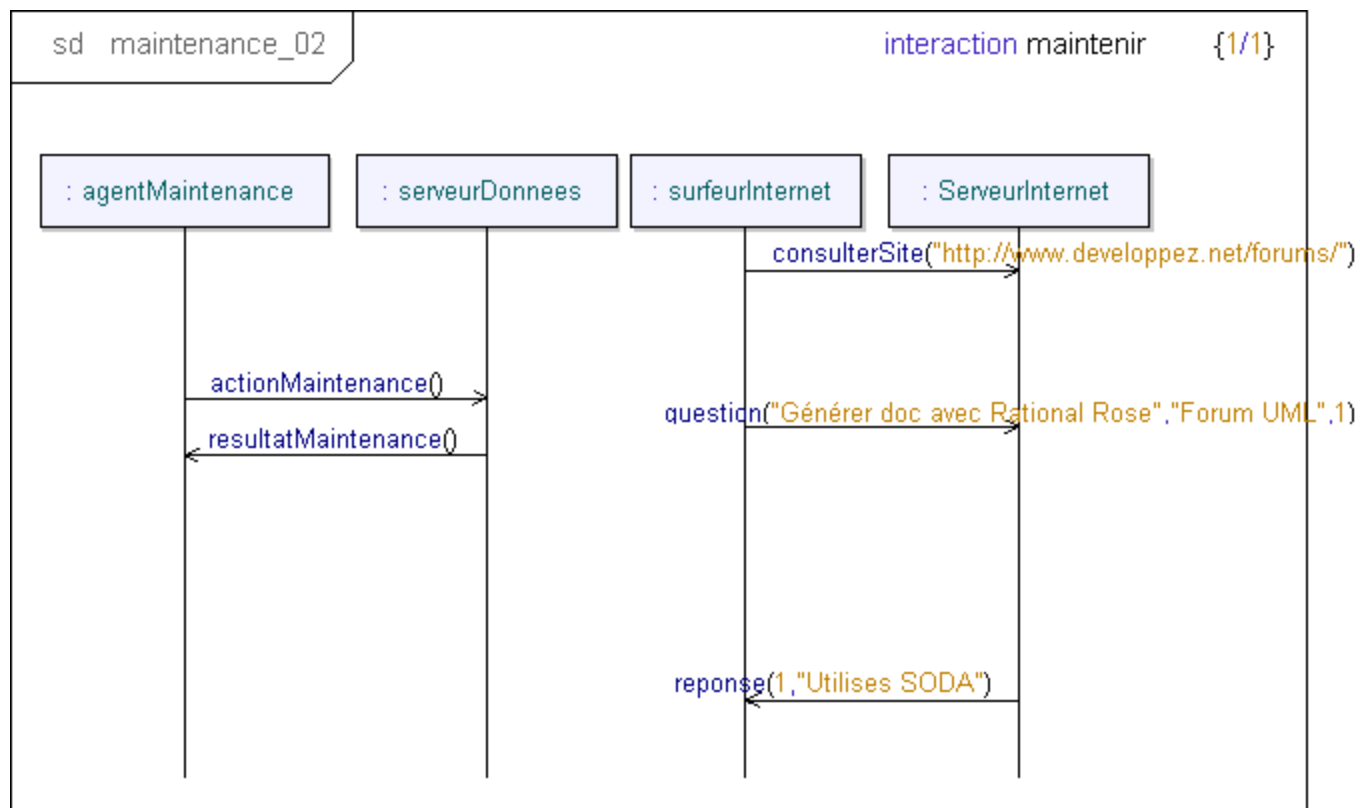
## Les fragments combinés

- La séquence faible : plusieurs parties chronologique
  - Au sein de chaque partie, la spécification existante de l'ordre de traitement
  - Les messages dans les différentes parties n'importe que
  - Les messages dans différentes parties de la même ligne de vie dans différents messages inclus chronologique
- Equivalent au parallélisme quand les messages inclus activent des ligne de vies distincts de celles activées dans les autres parties. Equivalent à la séquence stricte si une seule ligne de vie



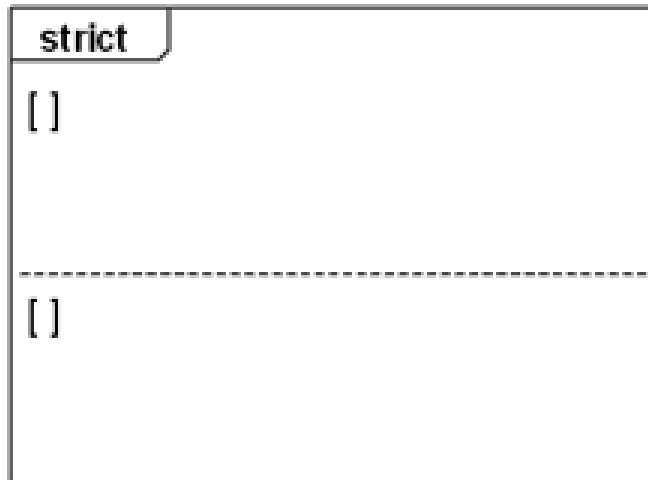


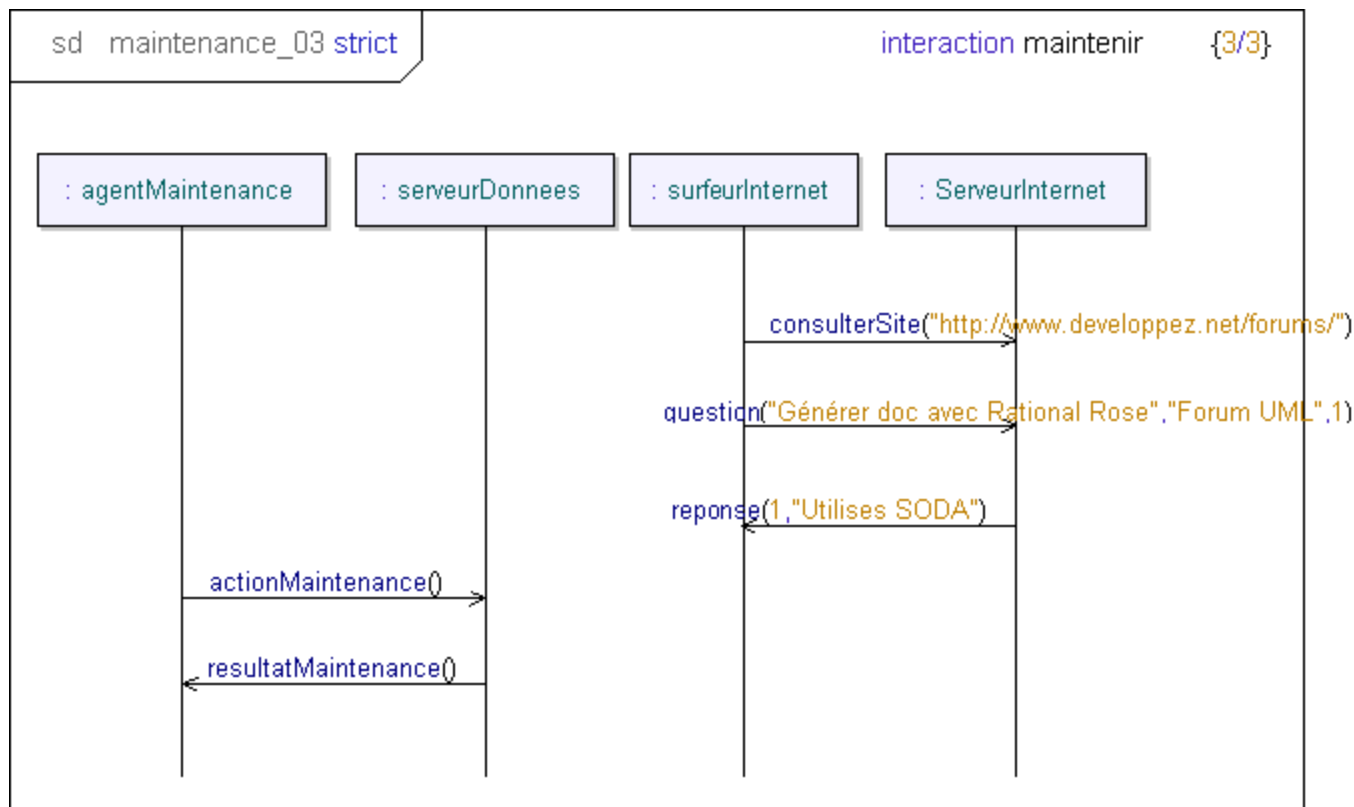




## Les fragments combinés

- La séquence stricte : Fragment composé de plusieurs parties introduites chronologiquement
  - Respect strict de l'ordre des messages

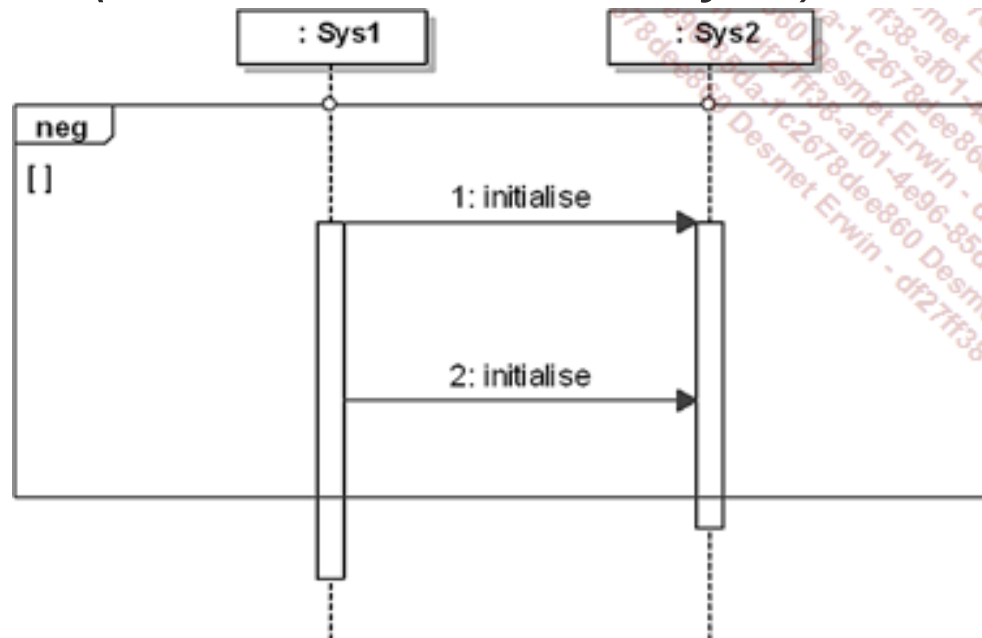




- La négation : décrit une séquence interdite

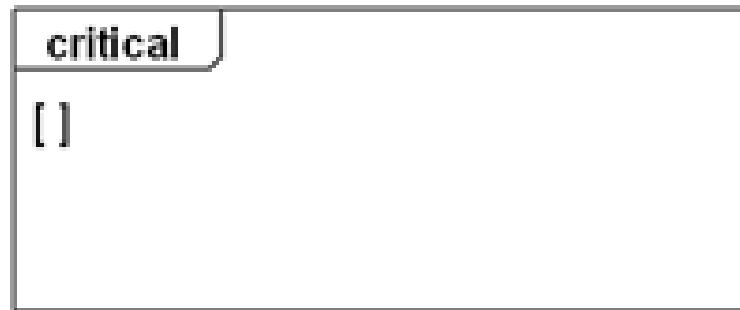


- Exemple : *Il est en effet interdit d'initialiser deux fois un même système (ici une instance de Sys2).*



## Les fragments combinés

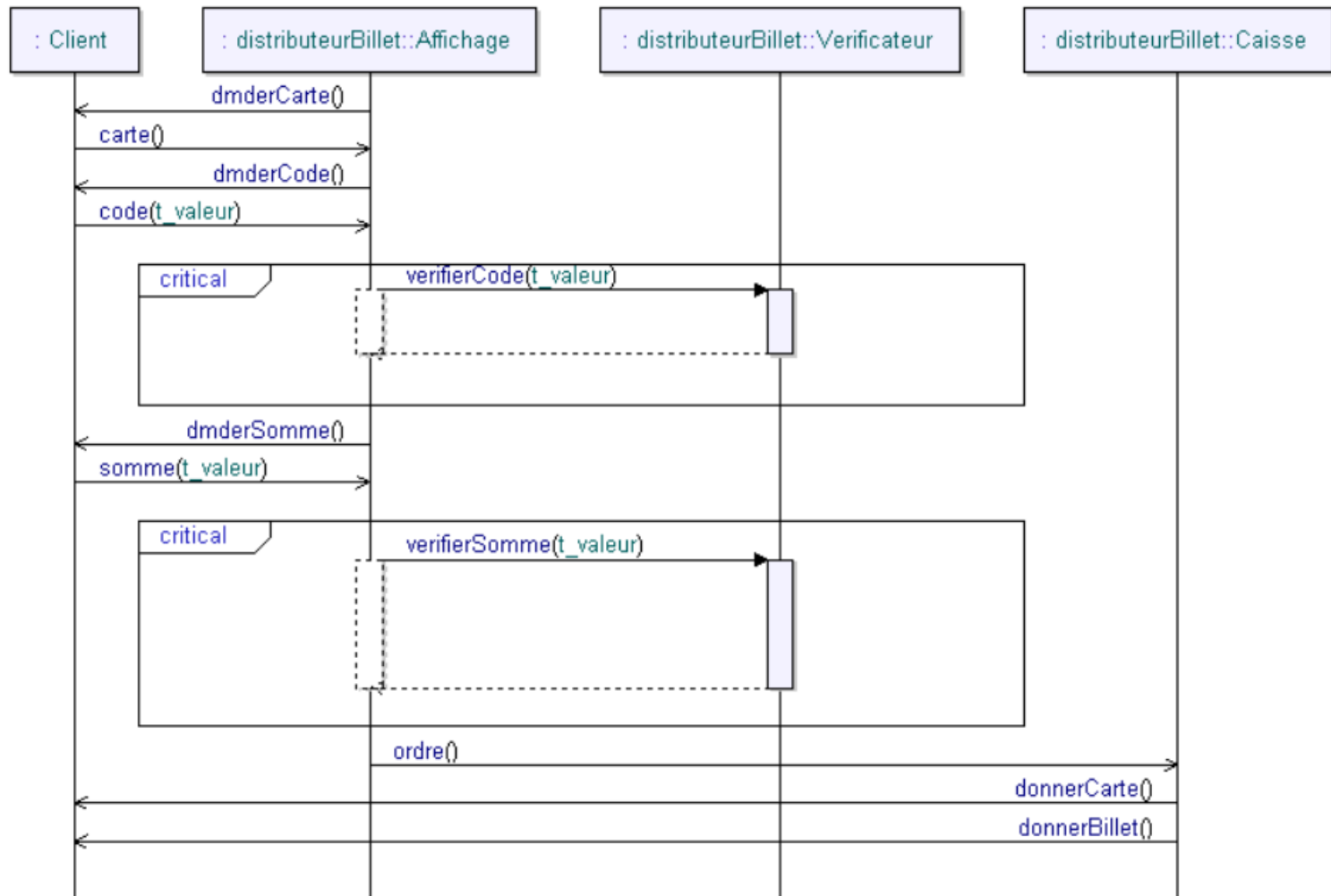
- La section critique : Ensemble de message qui doivent être entièrement traités avant l'acceptation d'autres.
  - Aucune interruption possible.
  - Pas de parallélisme,
- Chaque exécution de la section critique est dite atomique, c'est-à-dire qu'elle ne peut pas être divisée.



sd RetirerArgent\_critical

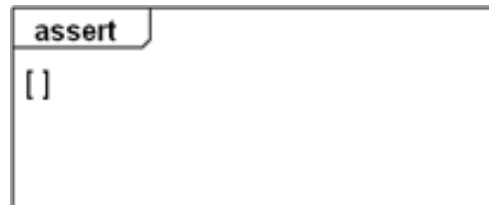
interaction retirerArgent

{6/6}



## Les fragments combinés

- L'assertion : peut-être le seul type d'envoi possible sans erreur par moment
- Il s'agit d'une notion proche des assertions dans les langages de programmation qui spécifient une condition sur l'état du système qui doit être vérifiée pour que cet état soit correct.

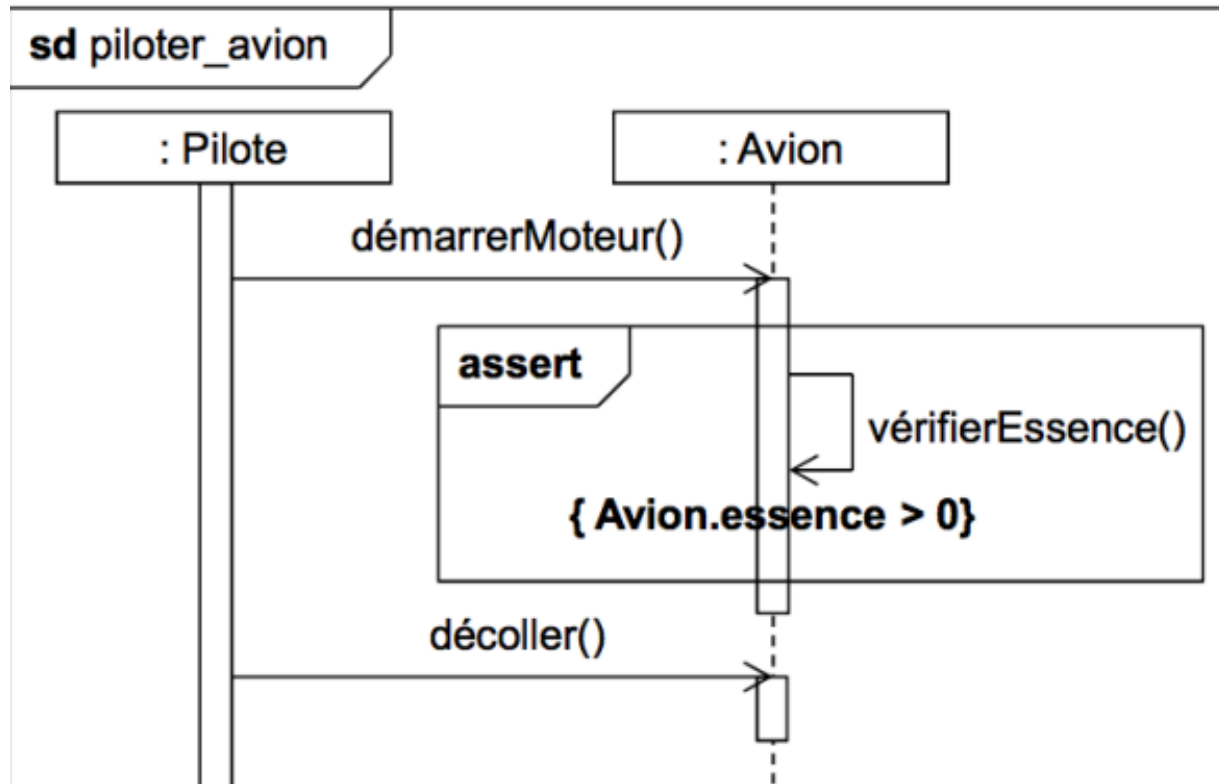


- *Une assertion peut indiquer qu'au début de l'exécution, il convient d'initialiser les différents objets d'un système en leur envoyant le message adéquat. Tout autre début d'exécution constitue une erreur.*



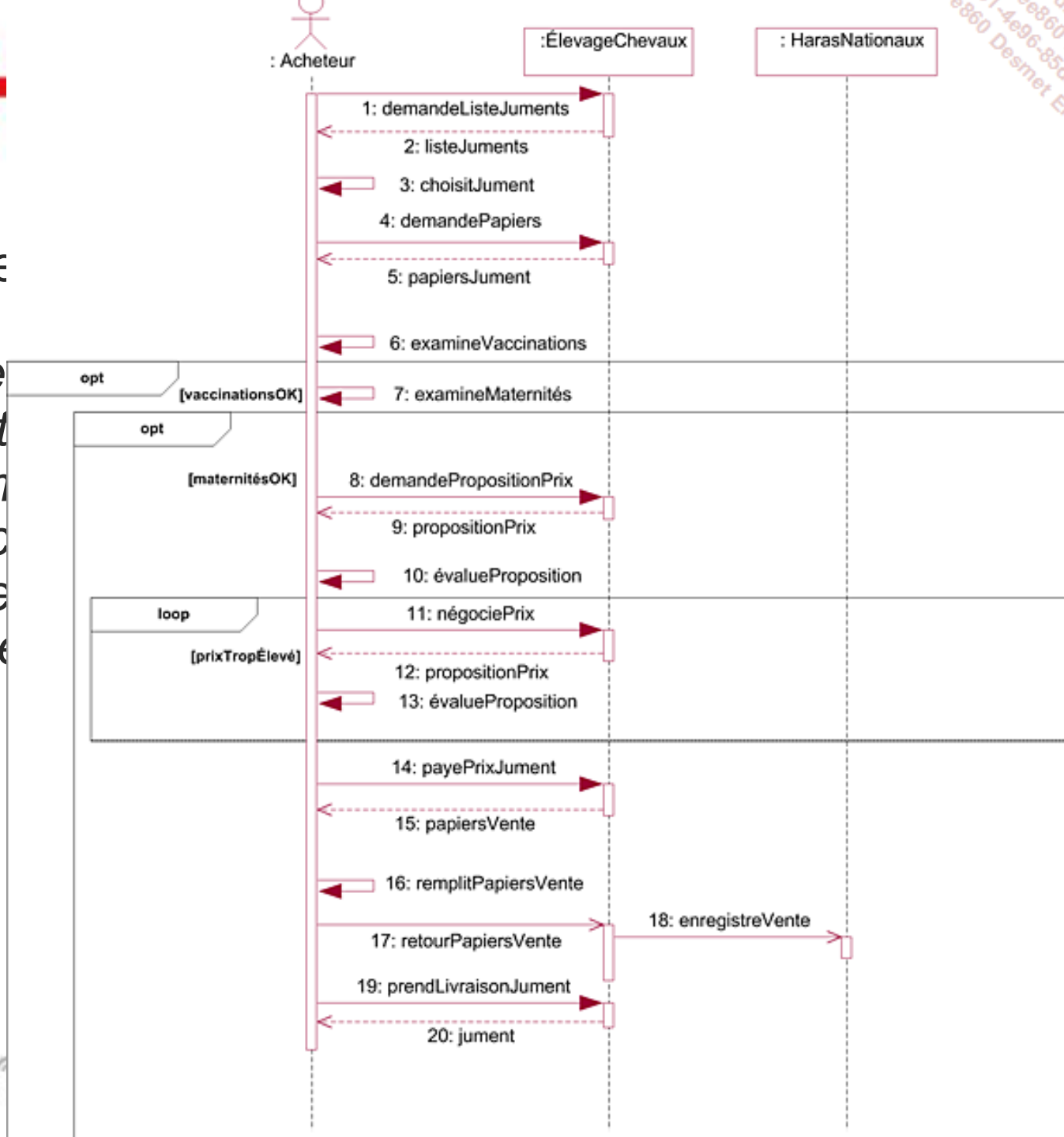
- **Rappel : qu'est-ce qu'une assertion**

Une assertion désigne, selon le Petit Larousse illustré, une proposition que l'on avance et que l'on soutient comme vraie : c'est donc l'équivalent d'une affirmation, la nuance résidant dans le fait (si mes souvenirs sont exacts) qu'une assertion n'est pas prouvée.



## Exemple

Les alternatives  
d'utilisation du  
diagramme de  
séquence sont  
les vaccinations.  
Par ailleurs,  
également



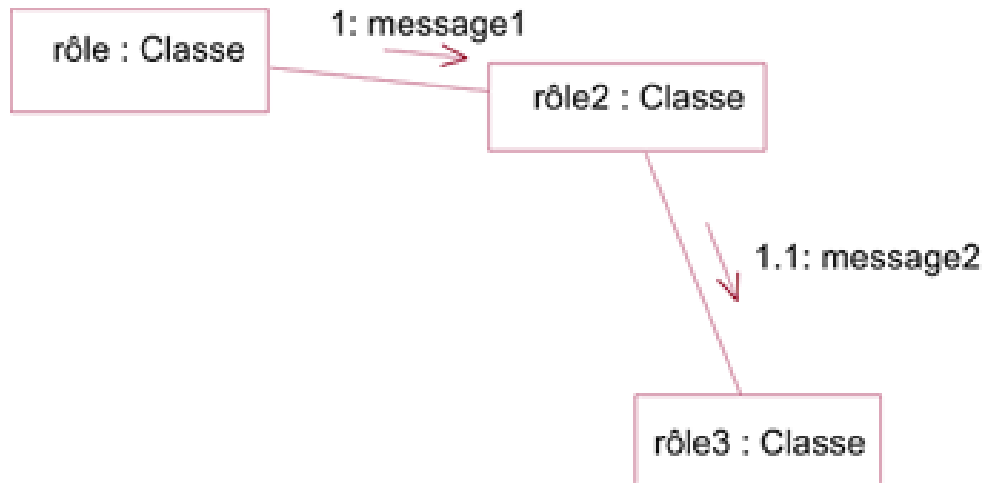
- [https://www.youtube.com/watch?v=-PBqaPVT91M&ab\\_channel=BouchraBouihi](https://www.youtube.com/watch?v=-PBqaPVT91M&ab_channel=BouchraBouihi)
- [https://www.youtube.com/watch?v=fKUn5f-kSAg&ab\\_channel=Prof.FatimazahraBARRAMOU](https://www.youtube.com/watch?v=fKUn5f-kSAg&ab_channel=Prof.FatimazahraBARRAMOU)
- <https://www.lucidchart.com/pages/fr/diagramme-de-sequence-uml>



- Alternative au diagramme de séquence
- Focus sur la représentation spatiale des objets
- Ici aussi les objets relié graphiquement aux objets avec qui il interagit
- Le cadre dans lequel apparaît l'objet est également appelé ligne de vie comme dans le diagramme de séquence.

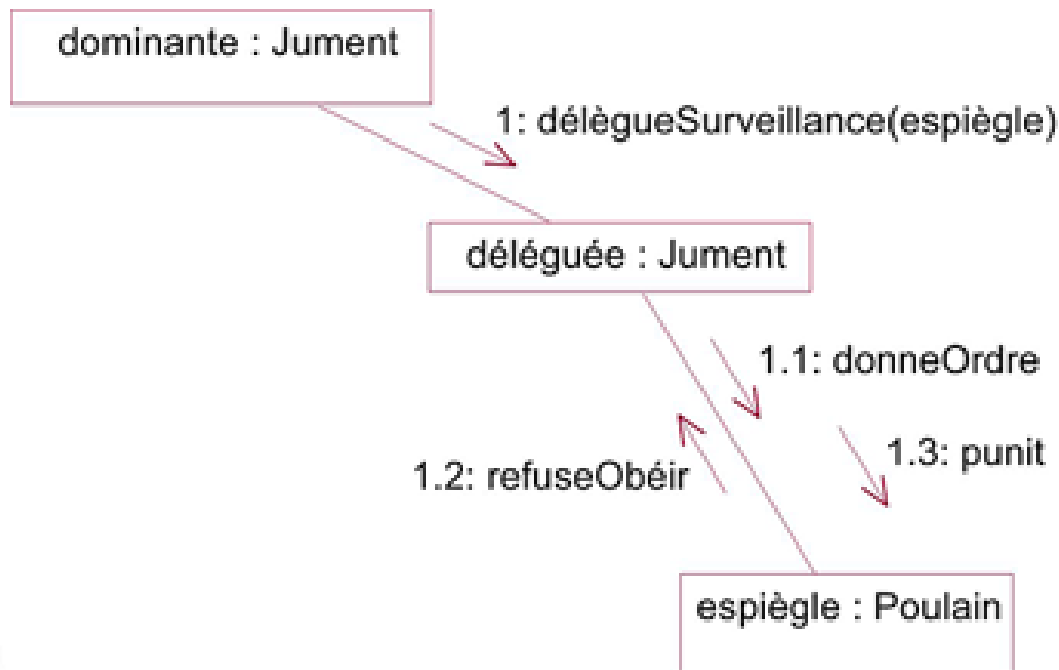
# Le diagramme de communication

- Message et ordre de ceux-ci
  - Les messages sont sur les liens inter-objets
  - Ils sont numérotés
  - Numérotation composée est possible
  - Paramètres et résultat sont possibles



## Diagramme de communication

- *Dans un troupeau de chevaux, il existe une jument dominante qui est responsable de l'éducation de tous les poulains. Elle peut passer le relais à une autre jument pour la surveillance d'un poulain particulier. La jument dominante délègue la surveillance du poulain Espiègle à une autre jument, qui lui donne un ordre auquel il refuse d'obéir. Il est donc puni.*

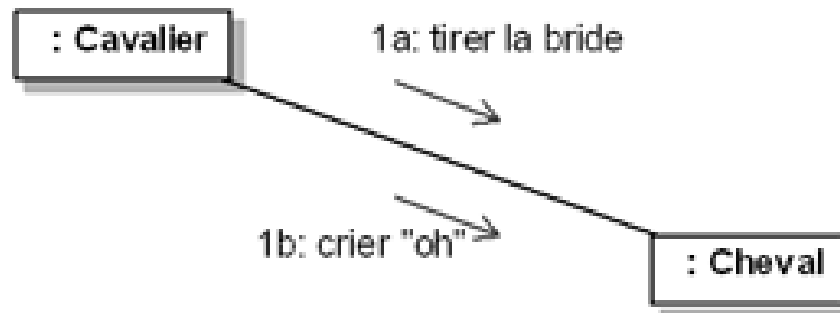




## Diagramme de communication

- Les messages parallèles : composition avec une lettre

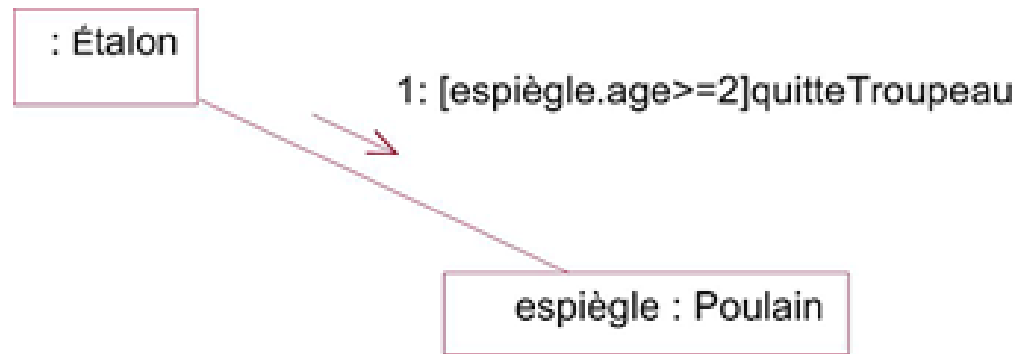
*Exemple : Il s'agit des ordres qu'un cavalier débutant envoie à son cheval pour l'arrêter alors qu'il court*



- Les messages itératifs :
  - Equivalent fragment combinés est inexistant
  - Mais boucle et test à l'envoi possible
- Mécanisme de test = condition entre crochets après le numéro: **Numéro[condition]: message**
- Mécanisme de boucle = on ajoute une \* :  
**Numéro\*[condition]: message**

## Diagramme de communication

Exemple : Un poulain est chassé du troupeau par l'étalon à 2 ans



L'âge est un attribut du poulain Espiègle. On y accède par la syntaxe **nomObjet.nomAttribut**. Cette syntaxe est recommandée pour l'accès aux attributs et aux méthodes d'un objet dans une condition.

- Messages itératifs et parallèles :
  - On ajoute deux | (barre verticale) après l'étoile

numéro\* || [condition]: message

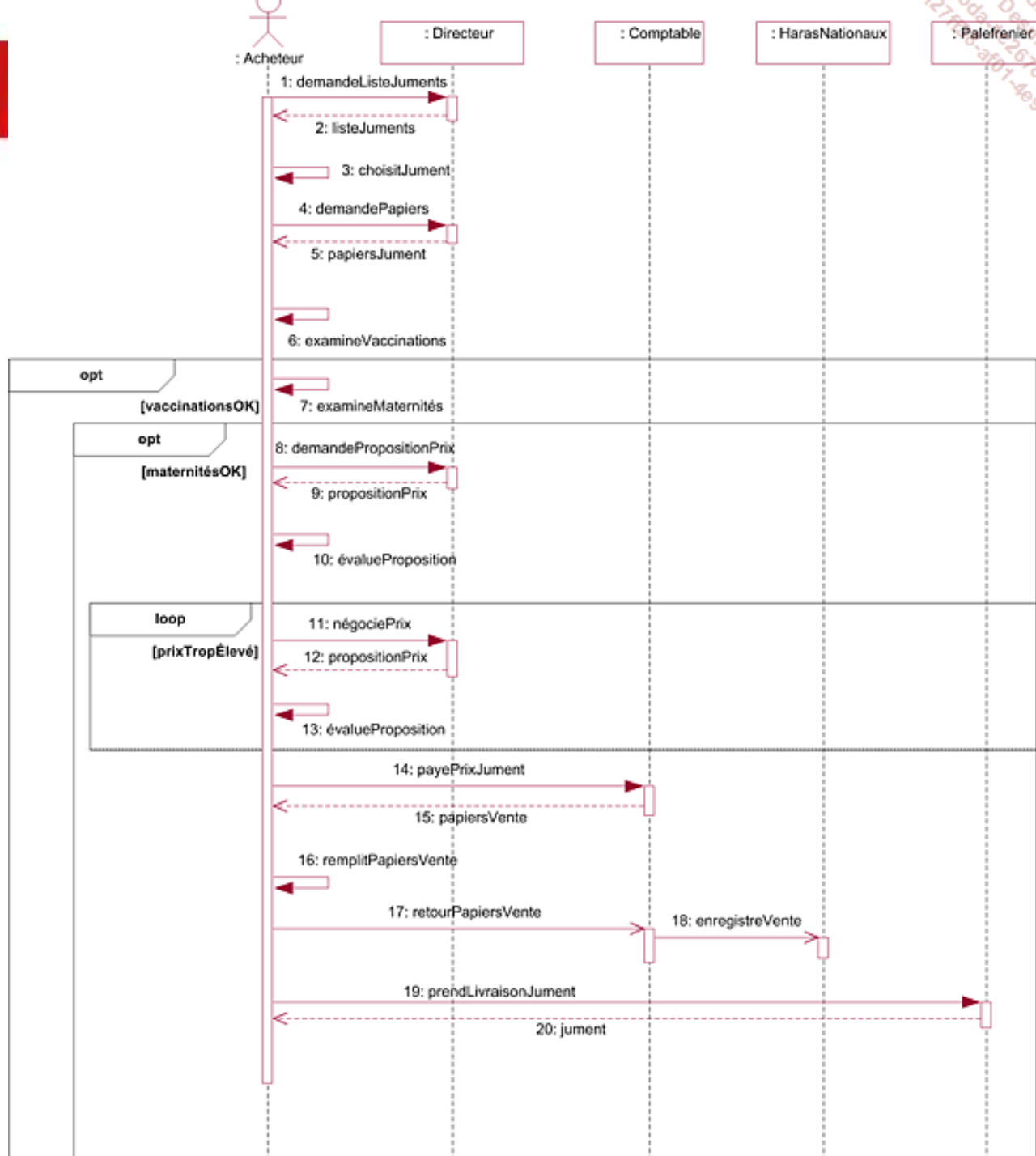
# Les objets du systèmes :



- 1 ) A quels objets du système sont destinés les messages?

Dans notre exemple précédent : on peut scinder en

- Acheteur
- Directeur
- Comptable
- Le palefrenier



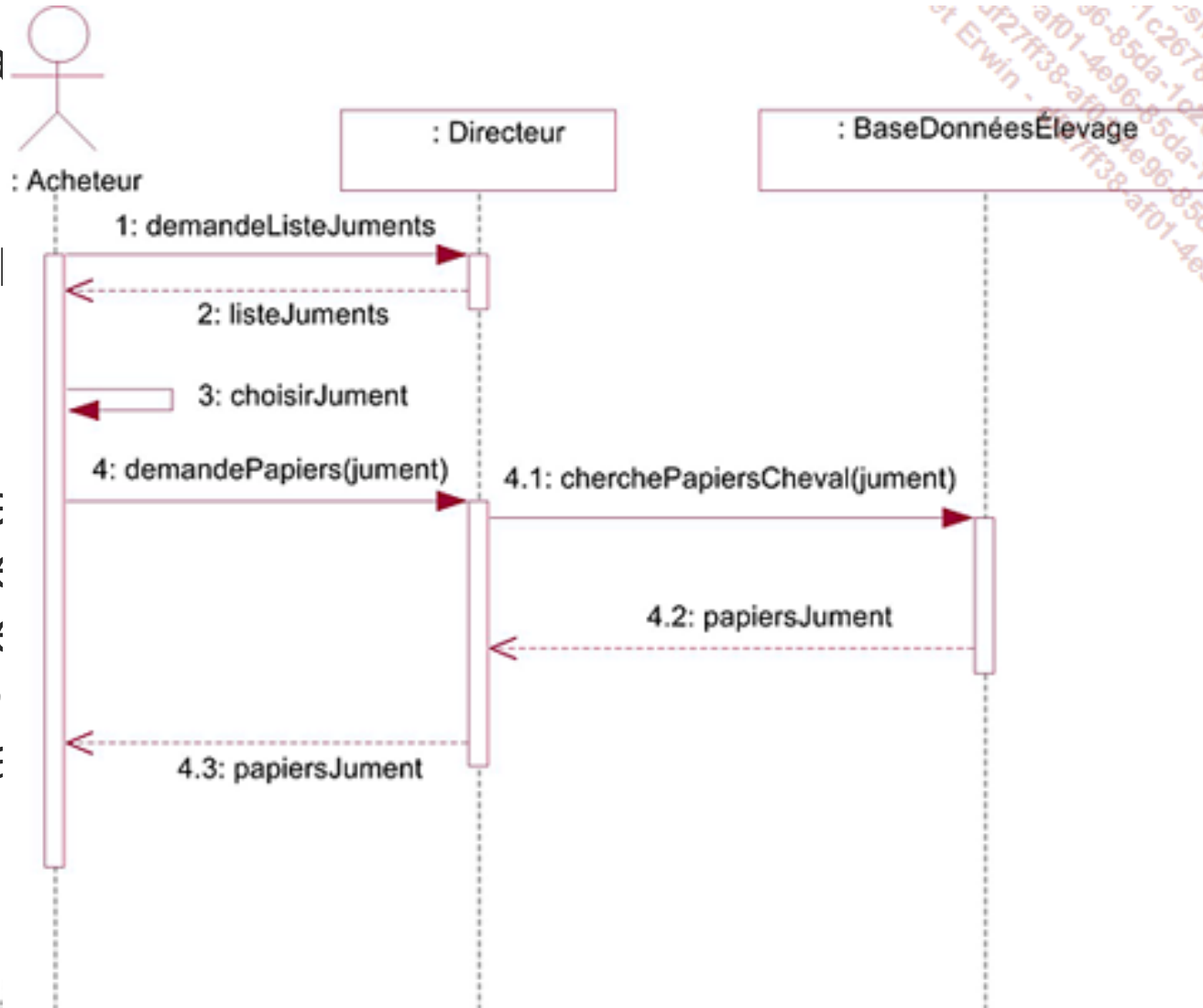


# Découverte des objets du systèmes

• 2) On va  
reçus

- Déco
- Enricl

Exemple :  
de la jume  
pour les r  
Au sein de  
transmise  
papiers d



papiers  
ge  
vant  
es

# Découverte des objets du systèmes

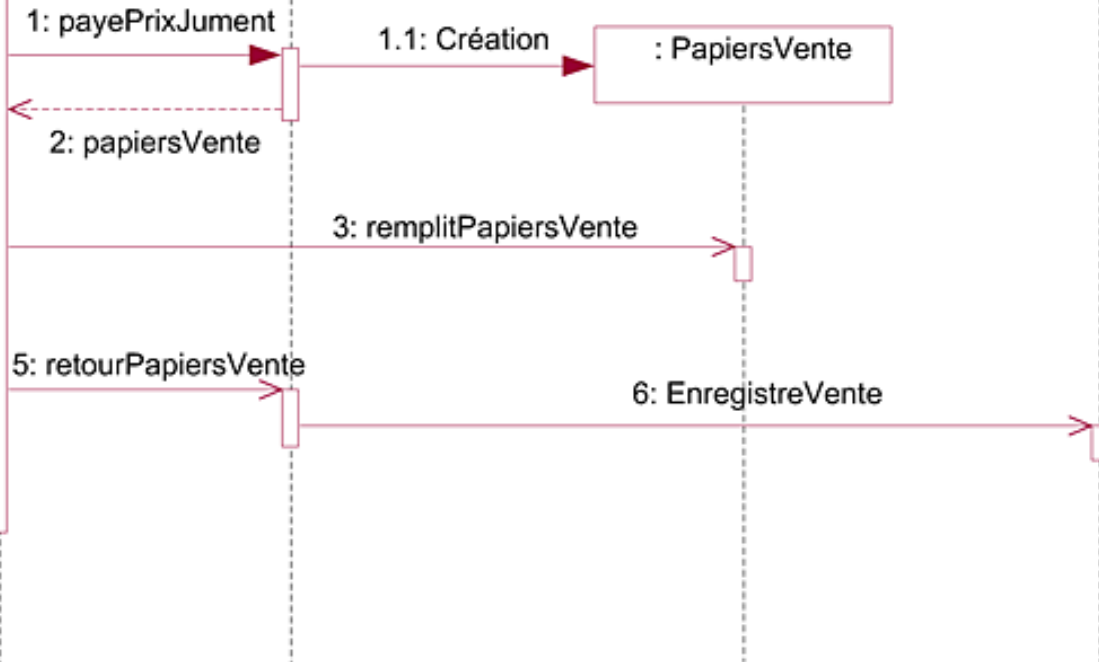
Lorsque  
vente a  
introduit

: Acheteur

: Comptable

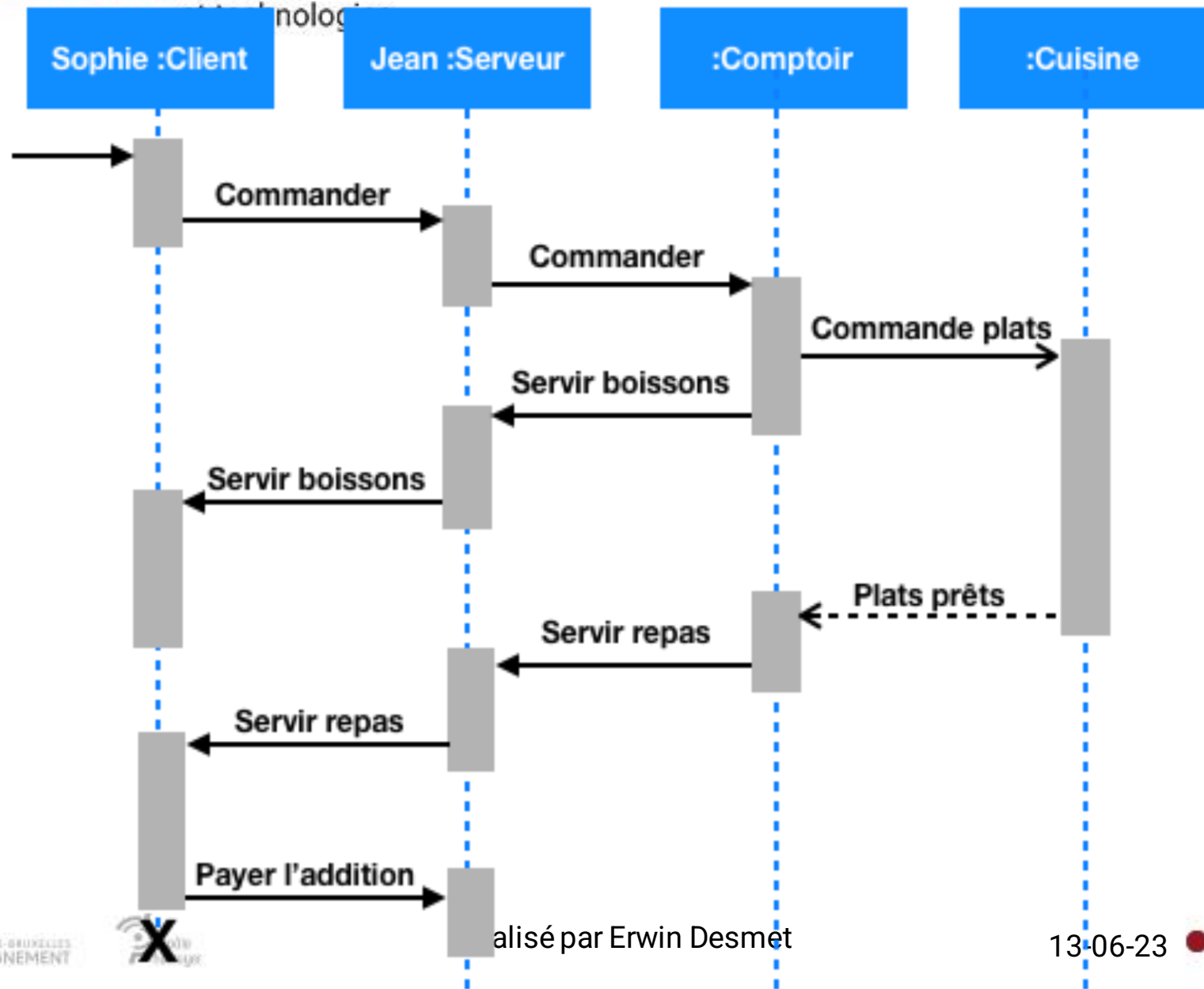
: HarasNationaux

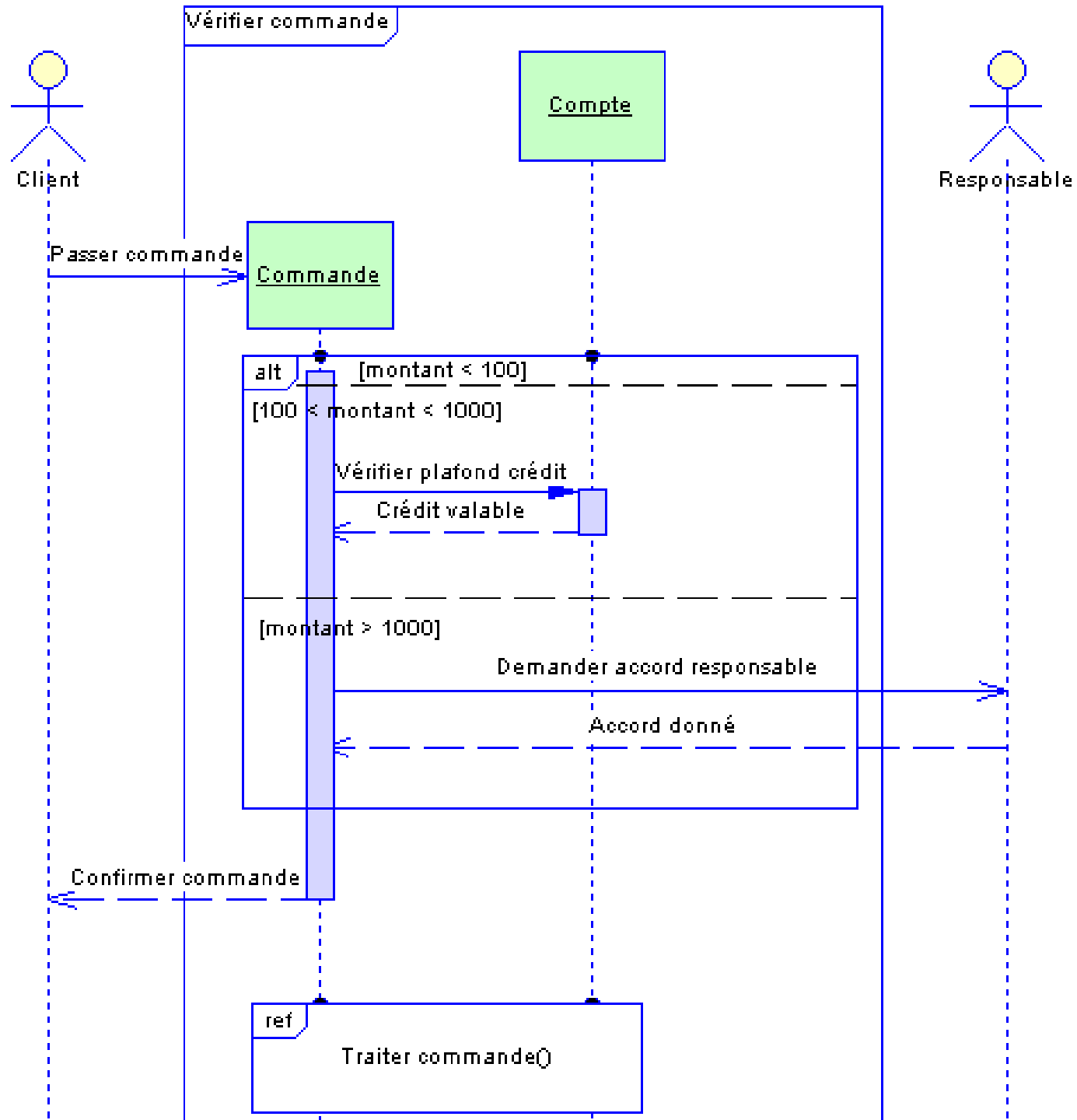
Papiers de la  
position  
vente



- Ce processus de décomposition des messages est itératif et doit être poursuivi jusqu'à l'obtention d'objets de taille suffisamment fine.
- On emploie le plus souvent le mot « grain » pour évoquer la taille d'un objet. En effet, il existe toute une panoplie de grains allant du plus fin au plus gros. Le système pris comme un seul objet est un objet de gros grain ou de granularité importante. À l'opposé, la base de données de l'élevage est déjà un objet de granularité plus fine. En son sein, une table ou une donnée seront des objets de granularité encore plus fine. Cette notion est relative. C'est à la personne en charge de la modélisation de déterminer le niveau de granularité des objets qu'elle désire obtenir.

- Les diagrammes de séquence et de communication sont importants pour les raisons suivantes :
  - Illustrer et vérifier le comportement d'un ensemble d'objets (système ou sous-système) ;
  - Aider à la découverte des objets du système ;
  - Aider à la découverte des méthodes des objets.
- Grâce aux fragments combinés, les diagrammes de séquence peuvent être utilisés pour décrire les cas d'utilisation.
- Les diagrammes de séquence mettent en avant les aspects temporels tandis que les diagrammes de communication montrent les liaisons entre les classes.





# Quizz

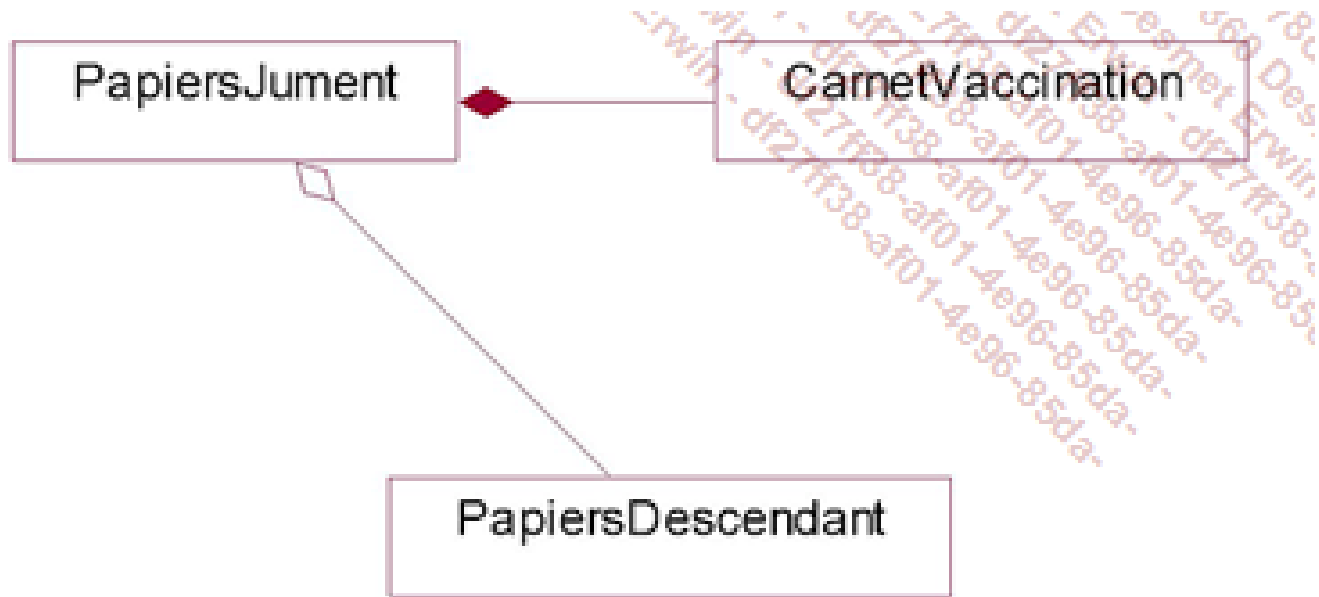
- Modélisation dite statique
  - Ne décrit pas les interactions
  - Ne décrit pas les cycles de vies
  - Méthodes sans descriptions de l'enchaînement
- Intro au diagramme de classe
  - Attributs/méthodes/associations d'objets
- Place central dans la conception de projet ! Il est OBLIGATOIRE (le seul)
- Approche OCL





- Découverte des objets dynamiquement dans la chap précédent (recherche les objets par apport aux messages)
- Autre approche : Décomposition de l'information d'un objet  
Cette info trop complexe pour être représenté par **un** objet

- *Dans l'exemple du chapitre La modélisation de la dynamique, le directeur recherche les papiers (dans le sens d'informations) de la jument à vendre dans la base de données de l'élevage. Cette base constitue un objet à grosse granularité composé lui-même d'autres objets, comme les papiers des chevaux, les informations financières et comptables, les documents d'achat et de vente de chevaux. Les papiers d'une jument sont composés, entre autres, de son carnet de vaccination et des papiers de ses descendants. Les papiers des descendants sont partagés par d'autres objets, comme les papiers de leur père étalon. Cette décomposition est guidée par les données et non par des aspects dynamiques.*

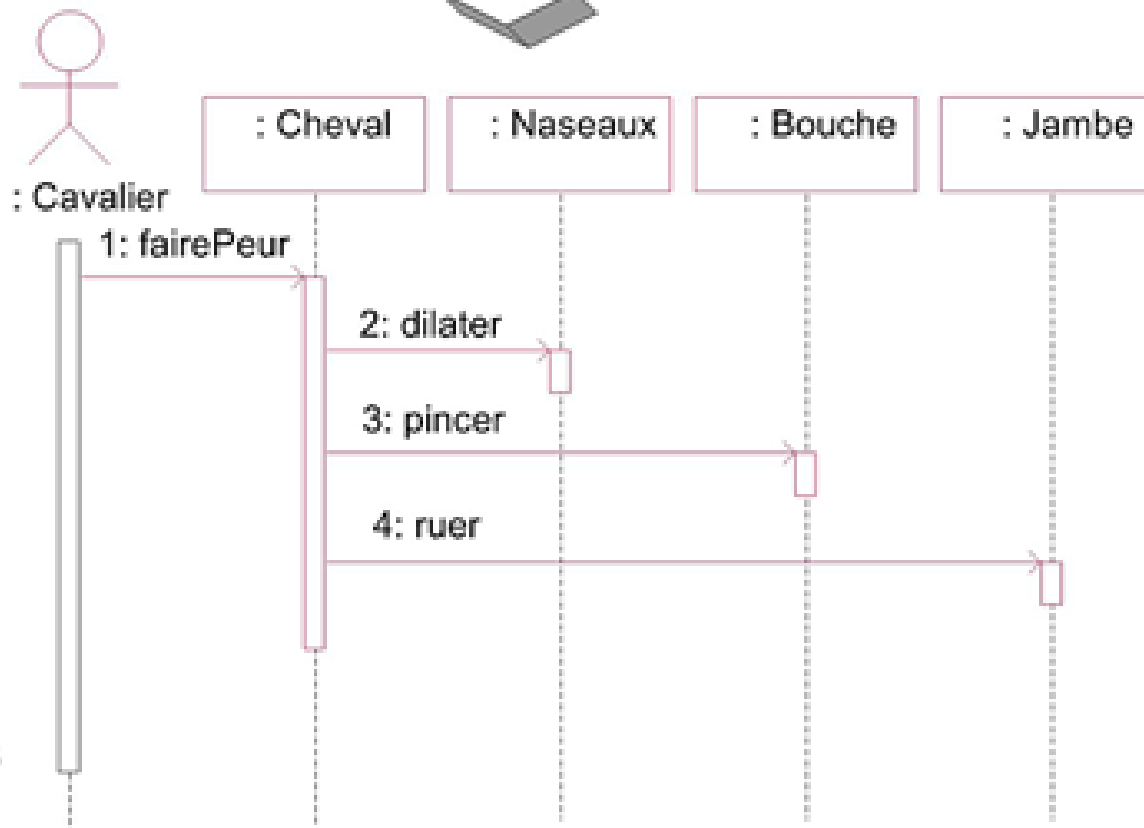
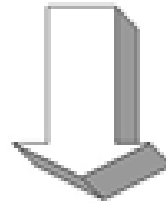
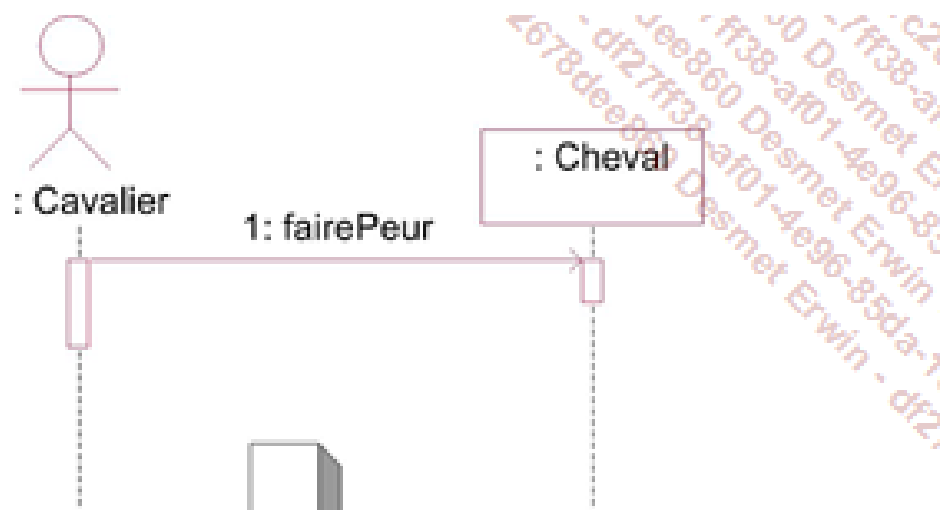


- Rappelons que la granularité d'un objet définit sa taille. Le système pris comme un objet est de gros grain ou de granularité importante. À l'opposé, le carnet de vaccination d'un cheval est un objet de grain beaucoup plus fin que le système.

## Partons d'une séquence

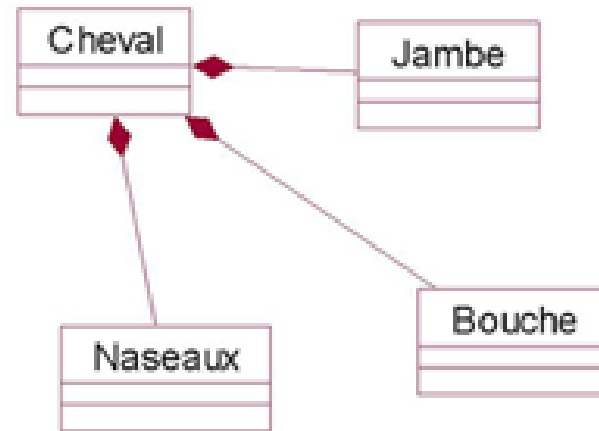
- Exemple 2 : *La décomposition d'un cheval pour faire apparaître ses différents organes peut se faire soit par la décomposition d'un diagramme de séquence, soit par la décomposition guidée par les données.*

*La décomposition par le diagramme de séquence consiste à analyser différents envois de message : faire peur, courir, manger, dormir. Ces derniers feront apparaître progressivement les différents organes du cheval. Illustrons le message fairePeur. Un cheval dilate ses naseaux pour marquer l'alerte, la surprise ou la peur. Il pince la bouche pour indiquer tension, peur ou colère. Enfin, la ruade est un mouvement défensif.*



## Depuis les données

- On va directement se concentrer sur les organes du cheval.
- Diagramme de classe lié :



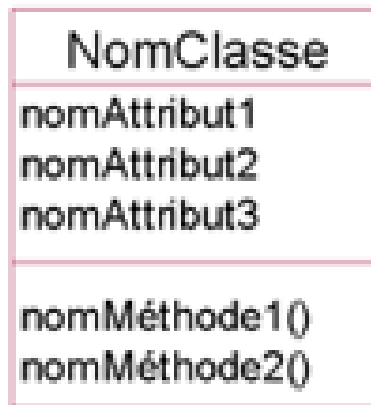
- Les deux approches apportent un résultat semblables
- Elles ne sont pas incompatibles
- La décomposition par données plus efficaces quand on connaît bien le domaine du système
  - Les objets sont souvent immédiat





## Représentation des classes

- La représentation UML est décomposé en 3 parties :
  - Le nom de la classe
  - Les attributs (informations portées par l'objet)
  - Les méthodes (services offerts par l'objet)



## Avant d'aller plus loin !

- Rappel :
- Le nom d'une classe est au singulier. Il est constitué d'un nom commun précédé ou suivi d'un ou plusieurs adjectifs qualifiant le nom. Ce nom est significatif de l'ensemble des objets constituant la classe. Il représente la nature des instances d'une classe.
- Le nombre d'attributs et de méthodes est variable selon chaque classe. Toutefois, un nombre élevé d'attributs et/ou de méthodes est déconseillé. Il ne reflète pas, en général, une bonne conception de la classe.

## Exemple forme simple : la classe Cheval

- Aucunes caractéristiques aux M et A
- 1<sup>er</sup> Phase de modélisation

Cheval
nom âge taille poids facultéVisuelle facultéAuditive
fairePeur() ruer() dilaterNaseaux() pincerBouche()

- Certains attributs et méthodes visible que de l'intérieur
- On parle de méthodes/attributs privés de l'objet
- L'attribut privé ou la méthode privée : la propriété n'est pas exposée en dehors de la classe, y compris au sein de ses sous-classes.
- L'attribut protégé ou la méthode protégée : la propriété n'est exposée qu'aux instances de la classe et de ses sous-classes.
- L'encapsulation de paquetage : la propriété n'est exposée qu'aux instances des classes de même paquetage. La notion de paquetage sera abordée plus tard

- Privé est rarement utilisée en réel
  - Instaure une différence entre les instances de classe et sous-classes
- Encapsulation de paquetage → Très liée à JAVA
- Conseil Protect

public	+	Élément non encapsulé visible par tous.
protégé	#	Élément encapsulé visible dans les sous-classes de la classe.
privé	-	Élément encapsulé visible seulement dans la classe.
paquetage	~	Élément encapsulé visible seulement dans les classes du même paquetage.



- Variable : attribut, paramètres, valeur de retour  
→ Tout élément pouvant prendre une valeur
- Le type est une contrainte appliqué à une variable
- Type spécifique comme une classe
- Type standard comme un entier (Integer), un réel(Float /Real), une chaine de caractères(String), un booléen(Boolean)



## Cheval

+ nom : String  
+ âge : Integer  
+ taille : Integer  
+ poids : Integer  
# facultéVisuelle : Integer  
# facultéAuditive : Integer

+ fairePeur()  
# ruer()  
# dilaterNaseaux()  
# pincerBouche()

- Une variable peut contenir plusieurs valeurs → Liste/Tableau
- On indique la cardinalité après le type : [borneInf..borneSup]
- Compris entre borneInf et borneSup
- On peut spécifier que un chiffre si il est exact
- \* peut-être utilisé comme borneSup
- [0..\*] peut être simplifié en \*

+ nom : String[1..3]

- Une propriété s'indique en accolades
  - {readOnly} : cette propriété indique que la valeur de la variable ne peut pas être modifiée. Elle doit être initialisée avec une valeur par défaut.
  - {redefines nomAttribut} : cette propriété n'est applicable qu'à un attribut. Elle spécifie la redéfinition de l'attribut de nom nomAttribut de l'une des surclasses. La redéfinition peut notamment porter sur le nom de l'attribut ou sur son type. En cas de changement de type, le nouveau type doit être compatible avec l'ancien, c'est-à-dire que son ensemble de valeurs doit être inclus dans l'ensemble des valeurs de l'ancien type.

- {ordered} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieure à 1), les valeurs doivent être ordonnées.
- {unique} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieure à 1), chaque valeur doit être unique (interdiction d'avoir des doublons). Cette propriété s'applique par défaut.
- {nonunique} : lorsqu'une variable peut contenir plusieurs valeurs (cardinalité supérieure à 1), la présence de doublons est possible.

+ nom : String[1..3]{unique, ordered}

- Les paramètres sont des valeurs transmises :
  - A l'appel : envoi du message appelant la méthode
  - Au retour d'appel de la méthode
- Résultat est une valeur transmise à l'objet appelant au retour
- Ils peuvent être typés
- L'ensemble : nom de la méthode, paramètres avec leurs noms, type, cardinalité, propriétés et type de résultat avec sa cardinalité et propriétés → signature de la méthode

nomMéthode (direction nomParamètre:

type[borneInf..borneSup]=ValeurDéfaut{propriétés}, ...):

typeRésultat[borneInf..borneSup]{propriétés}

- Rappel : Nbr de paramètres peut être nul  
type de résultat optionnel

On peut ajouter devant le nom du paramètres :

- in : la valeur du paramètre n'est transmise qu'à l'appel.
- out : la valeur du paramètre n'est transmise qu'au retour de l'appel de la méthode.
- inout : la valeur du paramètre est transmise à l'appel et au retour.
- Pas de mot clé = juste à l'appel

- Exemple : la classe **Cheval** dont la méthode **fairePeur** a été munie d'un paramètre, à savoir **l'intensité** avec laquelle le cavalier fait peur, et d'un retour qui est l'intensité de la peur que le cheval ressent. Ces deux valeurs sont des entiers. Quant aux autres méthodes, elles ne prennent pas de paramètres et ne renvoient pas de résultat.

Cheval
+ nom : String + âge : Integer + taille : Integer + poids : Integer # facultéVisuelle : Integer # facultéAuditive : Integer
+ fairePeur(intensite : Integer) : Integer # ruer() # dilaterNaseaux() # pincerBouche()

- Une instance contient une valeur spécifique pour chaque attributs
- Comment avoir des attributs comment ?
  - Attribut de classe : On souligne le nom dans le diagramme
  - Par habitude on donne une valeur par défaut !



*Exemple : Nous étudions un système qui est une boucherie exclusivement chevaline. Introduisons une nouvelle classe qui décrit un quartier de viande de cheval. Lorsque ce produit est vendu, il est soumis à une TVA dont le montant est le même pour tous les quartiers. Cet attribut est souligné et protégé car il sert à calculer le prix avec TVA incluse. Il est exprimé en pourcentage d'où son type Integer. La valeur par défaut est 210, soit 21 %, le taux de TVA des produits alimentaires en Belgique.*

*Currency implique des montants monétaires*

QuartierViandeChevaline
+ poids : Integer
+ qualité : Integer
# prixSansTVA : Currency
<u># tauxTVA : Integer = 55</u>
+ prixAvecTVA() : Currency

- Méthodes de classe : On envoie donc le message à la classe lui-même et pas aux instances.
- Ses méthodes ne travaillent qu'avec des attributs de classe

Exemple: *ajoute une méthode de classe à la classe **QuartierViandeChevaline** qui sert à fixer le taux de TVA. En effet, celui-ci est fixé par la loi et cette dernière peut être modifiée.*

QuartierViandeChevaline
+ poids : Integer + qualité : Integer # prixSansTVA : Currency <u># tauxTVA : Integer = 55</u>
+ prixAvecTVA() : Currency <u>+ fixeTauxTVA(nouveauTaux : Integer)</u>

- On parle de méthodes et attributs statique au lieu de classe en UML et par exemple en JAVA ou C++
- Ils ne sont pas hérités. En effet si il l'était, il y aurait autant d'attributs/méthodes que la classe n'a de sous-classes
- La sous-classes peut accéder aux attributs de la sur-classe sauf si encapsulation en privé !

## Les attributs calculés

- Valeur donnée par une fonction basée sur la valeur d'autres attributs.
- Il est précédé de / et suivi d'une expression de calcul

Exemple : *La méthode `prixAvecTVA` est remplacée par l'attribut calculé `/prixAvecTVA`*

QuartierViandeChevaline
+ poids : Integer
+ qualité : Integer
# prixSansTVA : Currency
# <u>tauxTVA</u> : Integer = 55
+ /prixAvecTVA : Currency = $(1 + \text{tauxTVA} / 1000) * \text{prixSansTVA}$

- Permet de lier des objets entre eux : Association
- Exemples
  - Le lien qui existe entre le poulain Espiègle et son père.
  - Le lien qui existe entre le poulain Espiègle et sa mère.
  - Le lien qui existe entre la jument Jorphée et l'élevage de chevaux auquel elle appartient.
  - Le lien qui existe entre l'élevage de chevaux Heyde et son propriétaire.
- Un lien est une occurrence de classe



## Lien entre les objets

- Une association relie des classes et porte un nom
- Chaque occurrence de cette association relie entre elles des instances de ces classes
- Exemples
  - L'association père entre la classe **Descendant** et la classe **Étalon**.
  - L'association mère entre la classe **Descendant** et la classe **Jument**.
  - L'association appartient entre la classe **Cheval** et la classe **ÉlevageChevaux**.
  - L'association propriétaire entre la classe **ÉlevageChevaux** et la classe **Personne**.

Les associations que nous avons examinées jusqu'à présent, à titre d'exemple, relient deux classes. De telles associations sont appelées associations binaires.

Une association reliant trois classes est appelée association ternaire. Une association reliant  $n$  classes est appelée association  $n$ -aire.

Dans la pratique, la très grande majorité des associations sont binaires et les associations quaternaires et au-delà ne sont quasiment jamais utilisées.



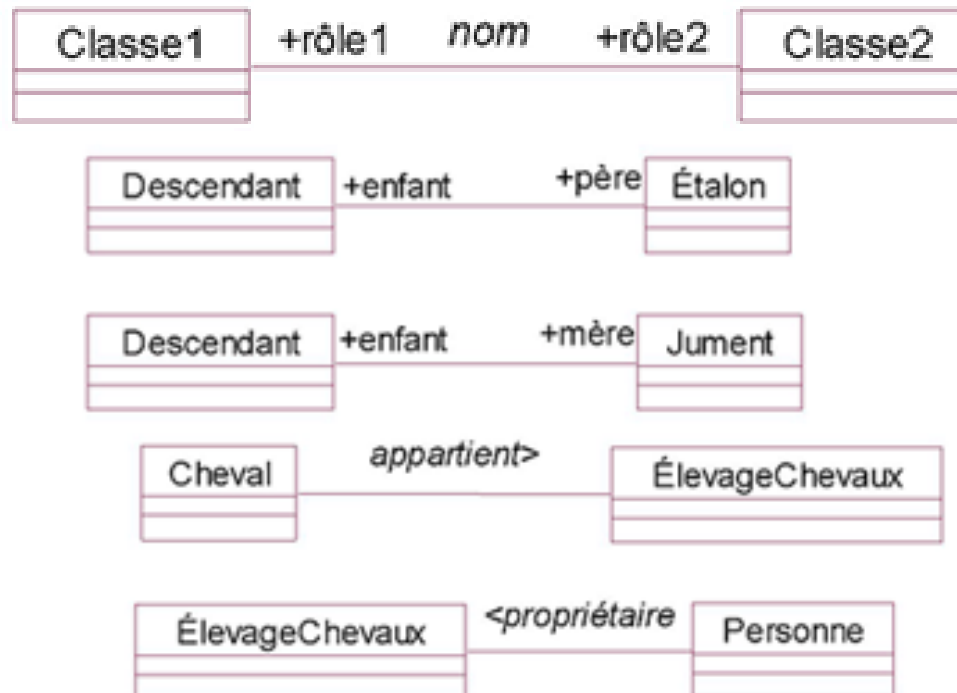
## Associations entre classes

- Association binaire : se représente par un trait continu dont elle associe les instances



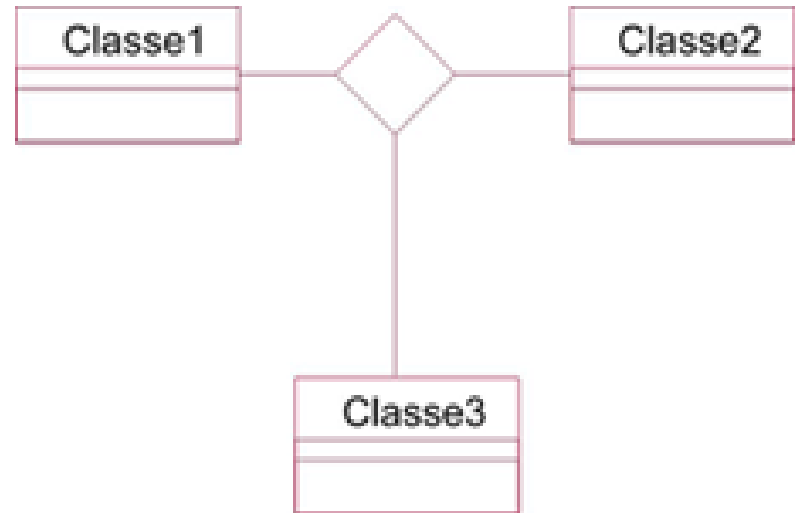
- Astuce : <, >, ^, v peuvent être mise sur les axes pour préciser le sens de lecture

- Nomination des extrémité des associations
  - Nom significatif du rôle que jouent les instances
  - Même nature qu'un attribut dont le type = classe à l'extrémité
  - Peut être privé, public, protect, ou de paquetage
  - On retire alors souvent le nom de l'association car similaire

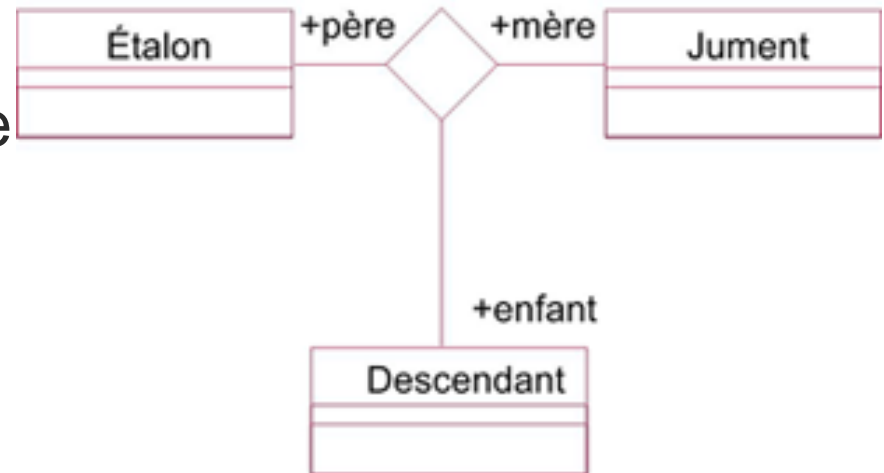


## Association ternaire

- Représenté par un losange



- Exemple : association famille



## Cardinalités des associations

- Se situe aux extrémités d'une association
- Indique le nombre d'instances de cette classe sont liés à la classe de l'autre extrémité
- En l'absence explicite de spécification, les cardinalités sont à 1

Spécification	Cardinalités
0..1	zéro ou une fois
1	une et une seule fois
*	de zéro à plusieurs fois
1..*	de une à plusieurs fois
M..N	entre M et N fois
N	N fois

- Exemple : ajoutons les cardinalités minimale et maximale de chaque association. Un élevage peut avoir plusieurs copropriétaires et une personne peut être propriétaire de plusieurs élevages.

À titre d'exemple, la première association se lit ainsi : un descendant possède un seul père, un étalon peut avoir de zéro à plusieurs enfants.



- Par défaut elle est bidirectionnelle
- Plus complexe à réaliser donc à éviter
- Possible d déterminer les liens de l'association depuis une instance de chaque classe d'origine
- Ajout d'une flèche pour aider les développeurs
- Exemple : *Dans le cadre particulier d'un élevage de chevaux, il est utile de connaître les chevaux que cet élevage possède, mais le contraire, à savoir connaître les élevages qui possèdent un cheval, n'est pas utile.*

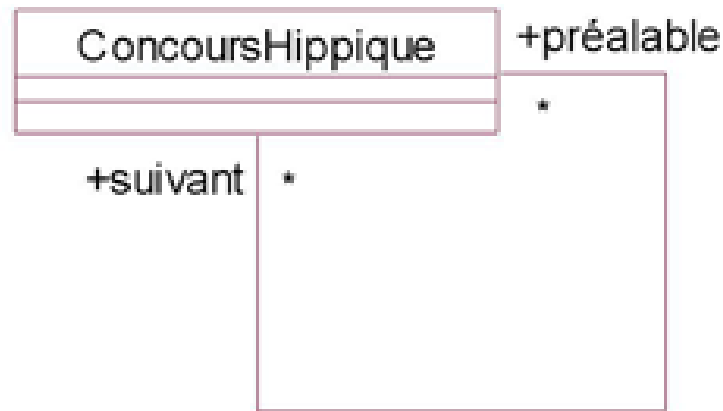


## Association réflexive

- La même classe sert d'entrée et de sortie
- On nommera le rôle de classe à chaque extrémité
- Sert principalement à décrire au sein des instances:
  - Des groupes d'instances
  - Une Hiérarchie au sein d'une instance
- Une fois expert on dira que le 1<sup>er</sup> cas = association d'équivalence et dans le 2eme cas = association d'ordre

## Association réflexive

*Exemple: Pour pouvoir passer les épreuves de sélection d'un concours hippique international, un cheval doit avoir gagné d'autres concours préalables. Il est donc possible de créer une association entre un concours et ses concours préalables. Cette association crée une hiérarchie au sein des concours.*

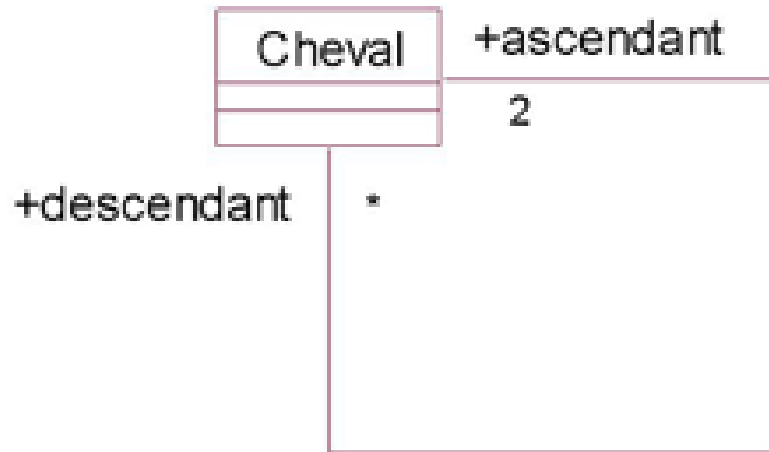




## Association réflexive

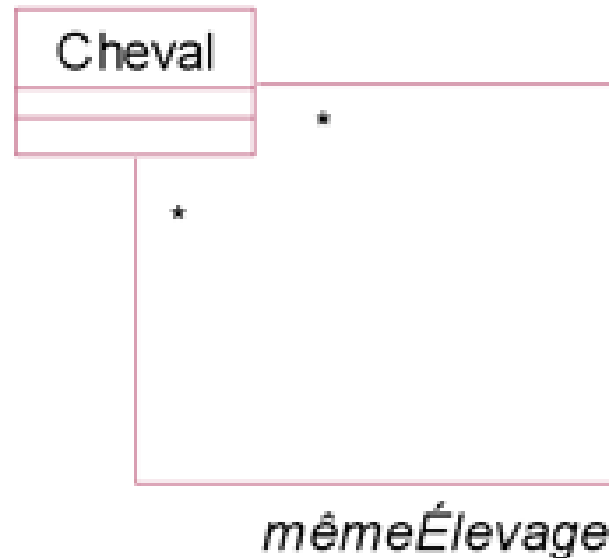
- *Exemple: l'association "ascendant/descendant direct" entre les chevaux. Cette association crée une hiérarchie au sein des chevaux.*

*Pour les ascendants, la cardinalité est 2 car tout cheval a exactement une mère et un père.*



## Association reflexive

- Exemple : *l'association entre les chevaux qui se trouvent dans le même élevage. Cette association crée des groupes au sein de l'ensemble des instances de la classe Cheval, chaque groupe correspondant à un élevage.*

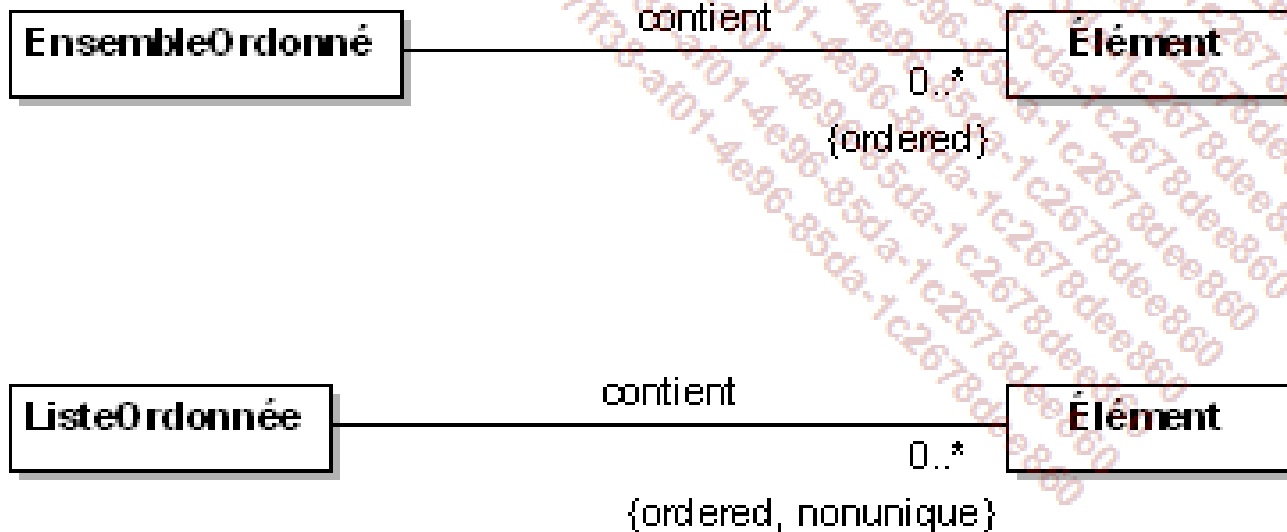


- Les extrémités des associations peuvent avoir des propriétés → comme un attribut
- Les principales :
  - {Ordered} : Lorsqu'une extrémité possède une cardinalité supérieure à 1, les occurrences doivent être ordonnées.
  - {nonunique}: cette propriété permet d'indiquer qu'une instance située à l'extrémité où se trouve la propriété peut être liée plusieurs fois à une instance située à l'autre extrémité. La cardinalité doit être supérieure à 1. Cette propriété n'est pas utilisée par défaut.

Exemple : *La partie supérieure montre la description d'un ensemble ordonné.*

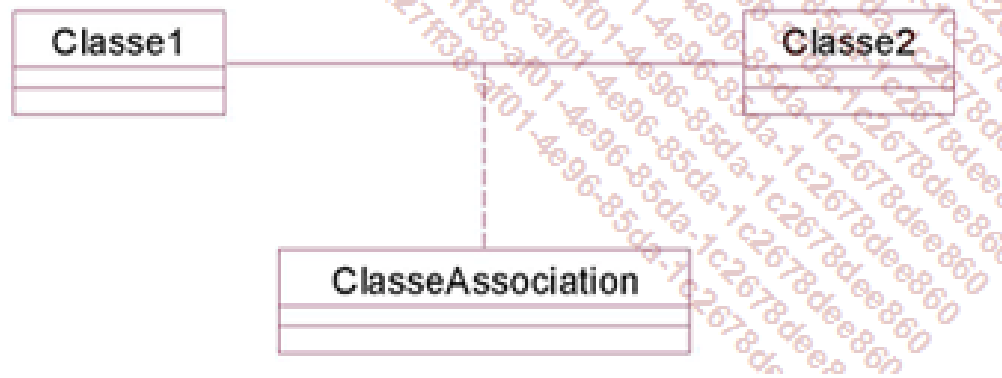
*Dans un ensemble, un même élément ne peut pas apparaître plus d'une fois. À l'opposé, dans une liste, un même élément peut apparaître plusieurs fois.*

*Dans la partie inférieure, une liste ordonnée est décrite. La propriété {nonunique} est donc utilisée.*

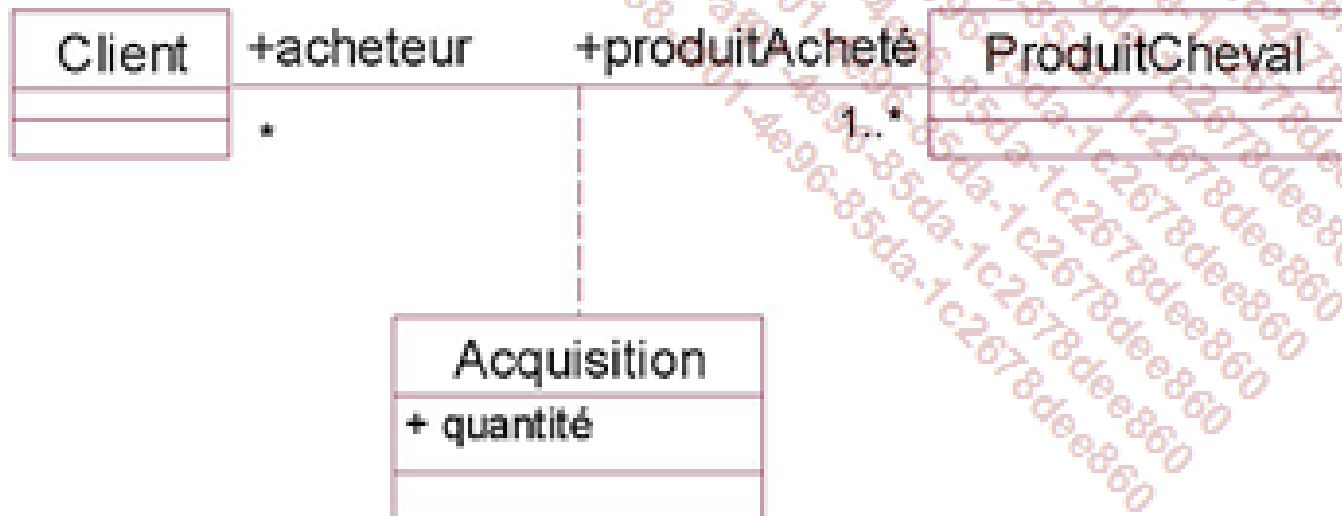


## Les classes-associations

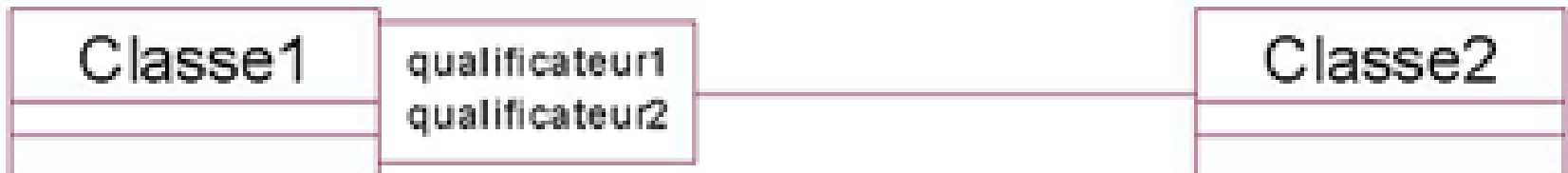
- Les liens entre les instances peuvent porter des infos
  - Elles sont spécifiques au lien
- L'association reçoit alors le statut de classe
- Ces instances sont donc des occurrences de l'association
  - Peut avoir des attributs, opérations et des associations
- On représente avec un trait pointillé



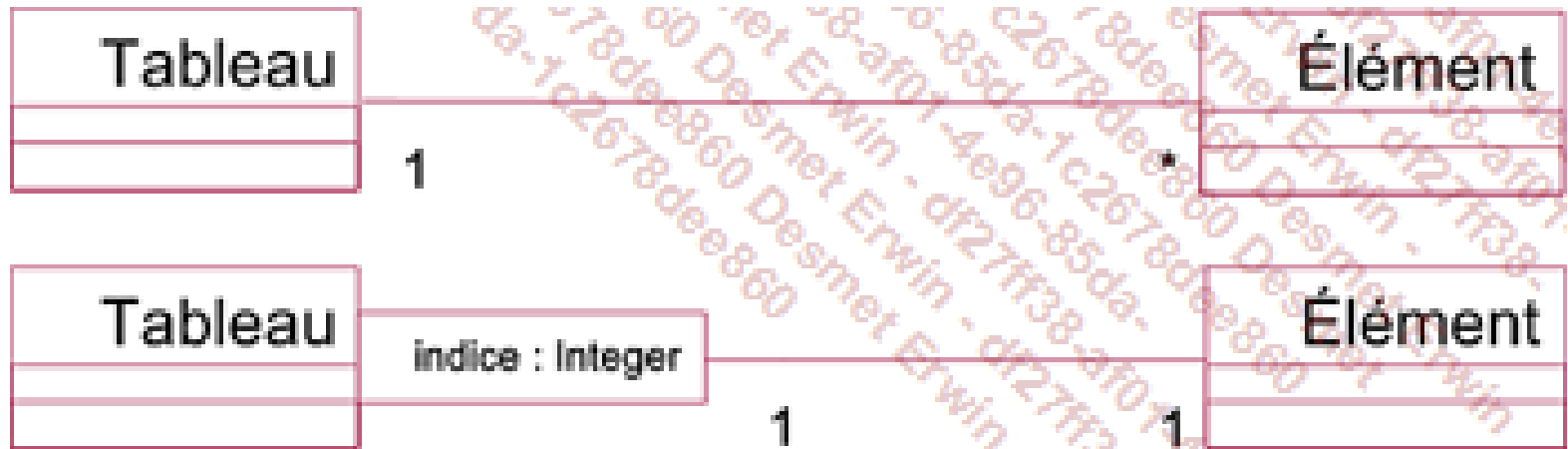
- Exemple : Un client achète un produit pour cheval (produits d'entretien, etc.), il convient de spécifier la quantité de produits acquis par une classe association, ici la classe Acquisition



- Cas de cardinalité max non finie mais quantifiables  
→ On peut passer la cardinalité à 1 avec cette option
- Valeur ou ensemble de valeurs qui ciblent l'instance
  - Souvent un index → par ex retrouvé indice dans un tab
- Inséré coté opposé ou est la cardinalité max
- Représenté par 1 ou plusieurs attributs



- Exemple : *Un tableau est constitué d'éléments. La figure décrit deux possibilités de modélisation en UML. La première ne fait pas appel à la qualification tandis que, dans la seconde, le qualificateur indice permet de retrouver un seul élément du tableau ; par conséquent, la cardinalité maximale passe à 1.*



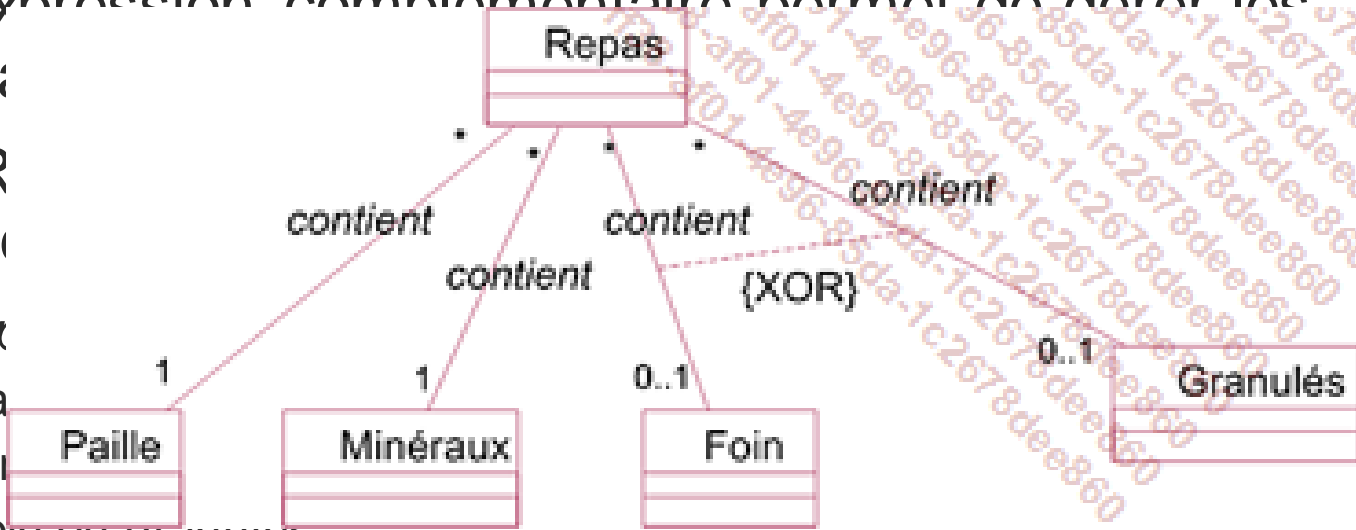


- Lors d'une course de chevaux, chaque cheval est numéroté. La figure illustre l'ensemble des participants d'une course en les qualifiant par leur numéro.



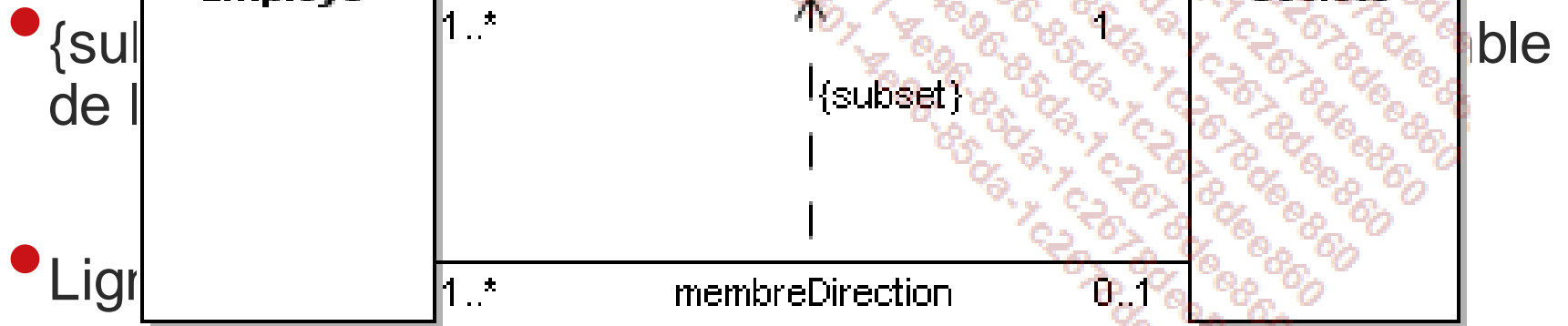
# Contraintes sur les associations

- Une expression complémentaire permet de créer les contraintes
- {XOR} classe les associations
- Exemple :
  - Paille
  - Minéraux
  - Foin ou granulés



La figure illustre le repas et ses différents constituants. Qu'il contienne soit du foin, soit des granulés est une contrainte exprimée par l'opérateur XOR. Celui-ci exprime la contrainte du ou exclusif entre les deux associations.

- Contraintes entre deux associations ayant les mêmes classes



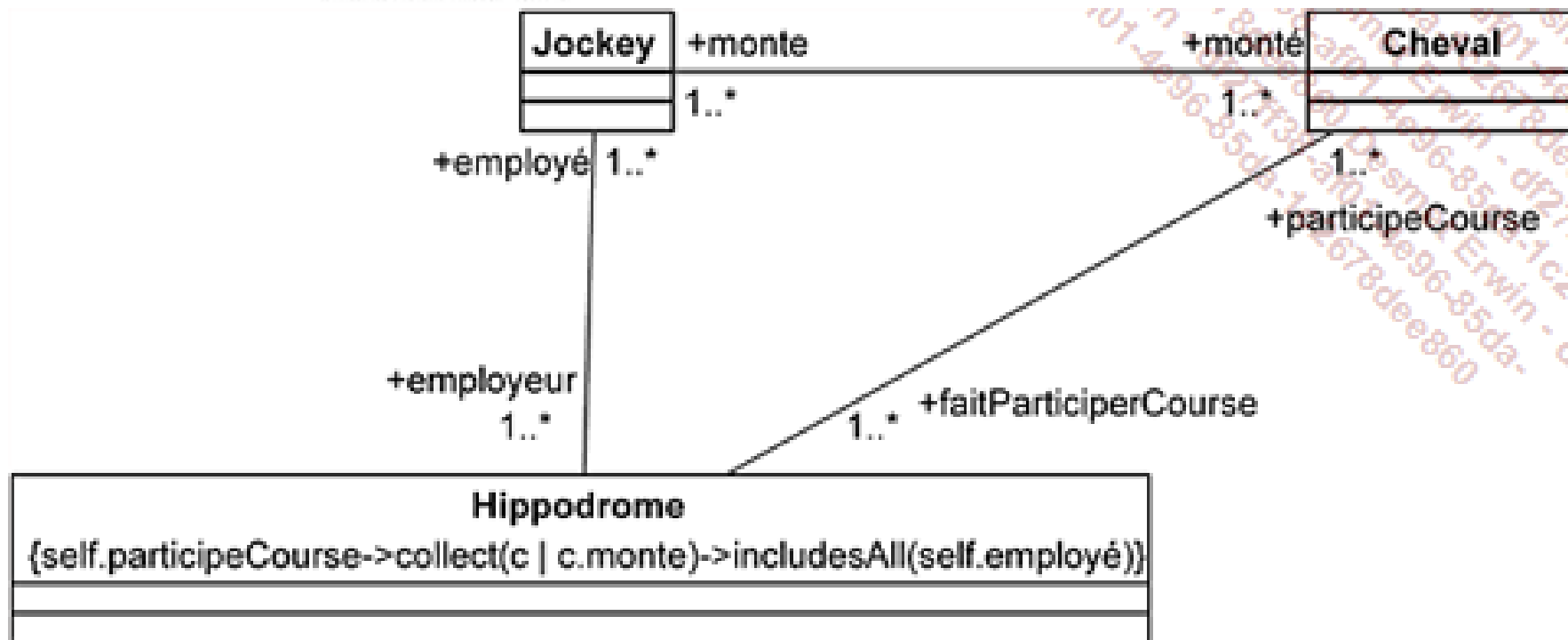
Exemple : *L'association membreDirection entre une société et ses employés est une association de type sous-ensemble de l'association membre entre les employés et ladite société.*

## Est-ce suffisant ?

- Et bien non ...
- Uml propose de s'appuyer sur OCL (ou langage naturel)
  - Langage sous forme de contrainte logique
  - On ajoute des notes aux diagrammes
  - Contraintes OCL toujours vrai ou fausse
  - Utilisation des opérateurs applicables sur des attributs par apport à leur type
  - On peut utiliser les méthodes des objets dans les conditions
  - Pour les ensembles OCL propose : **union, intersection, difference**
  - Pour les collections d'objets : **collect, includes, includesAll, asSet, exists, forAll**

- Pour utiliser une méthode ou un attribut → Self
- Exemple :
  - **self.attribut**
  - **self.methode**
  - **self.role.role. ....role.attribut**
  - **Self.role.role....role.methode**
- Si pas directement dans le diagramme de classes, préciser le contexte
  - **Context Classe inv Contrainte :**

Source OCL : The Object Constraint Language : Getting Your Models ready for MDA, Second Edition de Jos Warmer et Anneke Kleppe, Addison-Wesley, 2003.



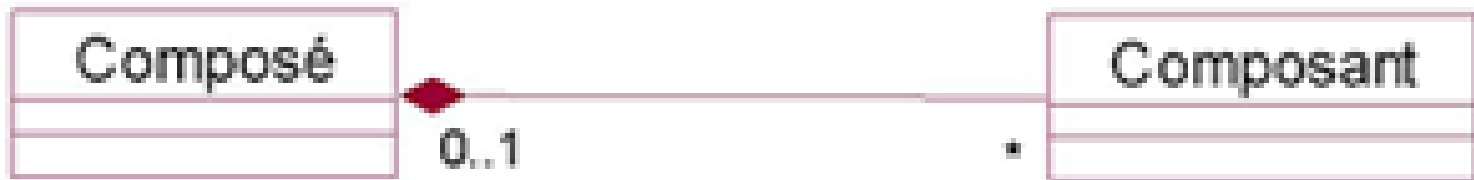
**Context** Hippodrome **inv** jockeysEmployés:

**self**.participeCourse->collect(c | c.monte)->includesAll  
(**self**.employé)

## Les objets composés

- Un objet composé d'autres objets
- On parle d'association de composition
- Type faible ou fort

- Les composants sont une partie de l'objet composé
- Un composant ne peut appartenir qu'à un composé
- La suppression du composé entraîne la suppression des composants



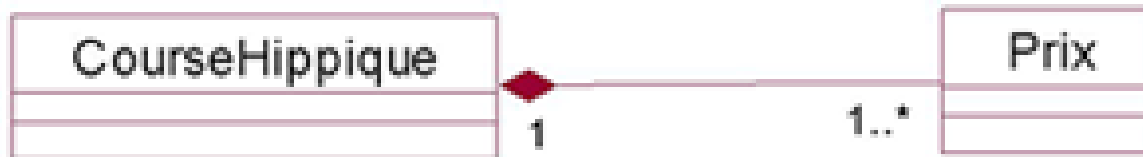
- Nous prendrons comme norme d'appeler composition forte : composition



*Exemple : Un cheval est composé d'un cerveau. Le cerveau n'est pas partagé. La mort du cheval entraîne la mort de son cerveau. Il s'agit donc d'une association de composition*



*Exemple : Une course hippique est constituée de prix. Ces prix ne sont pas partagés avec d'autres courses (un prix est spécifique à une course). Si la course n'est pas organisée, les prix ne sont pas attribués et ils disparaissent. Il s'agit d'une relation de composition.*



- On l'appelle agrégation.
- Un composants peut être partagés à plusieurs composés
- La destruction du composé n'implique pas la fin du composants
- Est plus fréquente
- En modélisation, en phase1 on mets souvent que des agrégation (sauf certitude)

## Les objets composés : Rappel

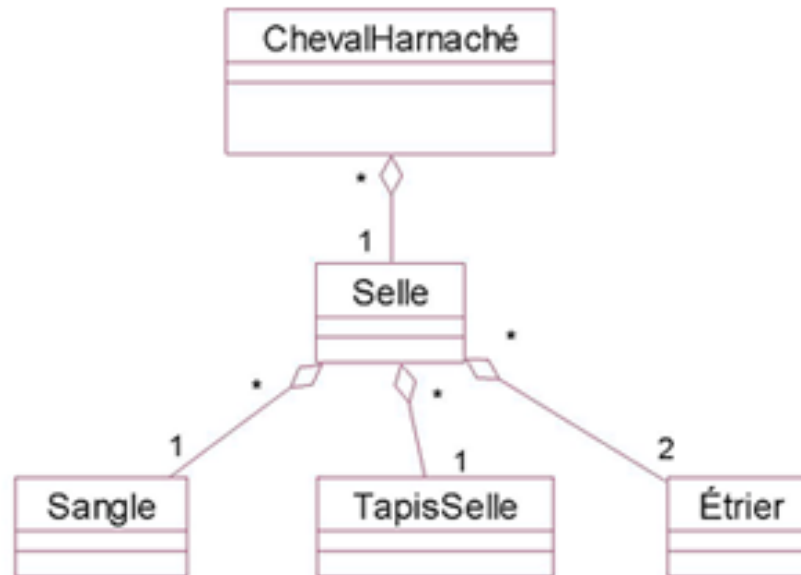
Déterminer, sur un modèle, qu'une association d'agrégation est une association de composition, revient à ajouter des contraintes, au même titre que donner un type ou préciser des cardinalités.

Nous avons étudié les contraintes en UML, en langage naturel ou en OCL.

Ajouter des contraintes, c'est ajouter du sens, de la sémantique à un modèle, c'est l'enrichir. Il est donc normal que ce processus d'enrichissement requière des phases successives.

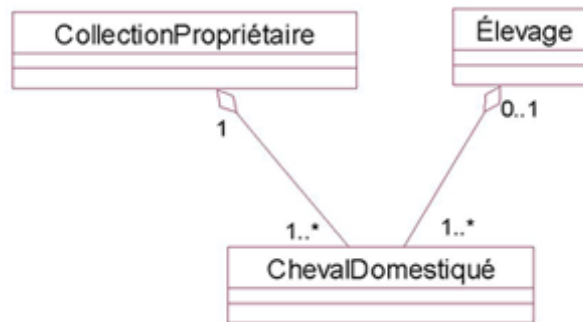
## Les objets composés : agrégation

Exemple : *Un cheval harnaché est composé d'une selle. Une selle est elle-même composée d'une sangle, d'étriers et d'un tapis de selle. Cette composition relève de l'agrégation. En effet, la perte du cheval n'entraîne pas la perte de ces objets et la perte de la selle n'entraîne pas la perte de ses composants.*



## Les objets composés

Exemple : *Un propriétaire équestre possède une collection de chevaux. Un cheval domestiqué appartient à une seule collection et peut simultanément être confié ou non à un élevage. Il peut être alors composant des deux agrégations*



*De façon plus précise, un cheval peut appartenir à plusieurs propriétaires. Dans ce cas, la cardinalité au niveau de la collection du propriétaire n'est plus 1 mais 1..\* pour exprimer la multiplicité. Le cheval peut être alors partagé plusieurs fois dans une même agrégation*



## Les objets composés : résumés

	Agrégation	Composition
Représentation	losange transparent	losange noirci
Partage des composants par plusieurs associations	oui	non
Destruction des composants lors de la destruction du composé	non	oui
Cardinalité au niveau du composé	quelconque	0..1 ou 1

# La dépendance

- Une classe à besoin d'une autre
- Représente par une flèche pointillé allant vers la classe du besoin

Exemple : La classe **Client** possède une méthode dont le type de l'un des paramètres ou le type de retour est la classe **Fournisseur**.



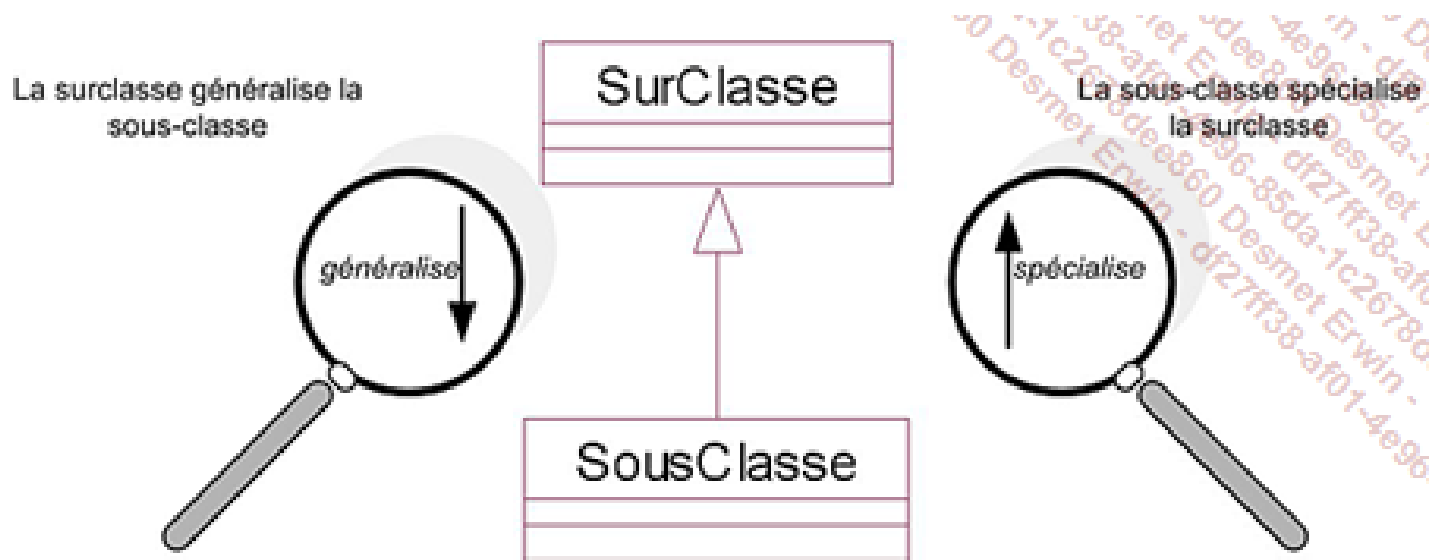


- On peut ajouter un stéréotype pour définir la sémantique
- «**call**» : l'implantation de la classe Client invoque une méthode de la classe Fournisseur.
- «**create**» : l'implantation de la classe Client crée une instance de la classe Fournisseur pour un usage interne.
- «**derive**» : la spécification et l'implantation de la classe Client sont uniquement obtenues à partir de la spécification et de l'implantation de la classe Fournisseur. Le client est donc une redondance qui a été construite, par exemple, pour des raisons d'optimisation ou de facilité d'utilisation.
- «**use**» : il s'agit du stéréotype plus général qui spécifie que la classe Client a besoin, sans fournir plus de précisions, de la classe Fournisseur.

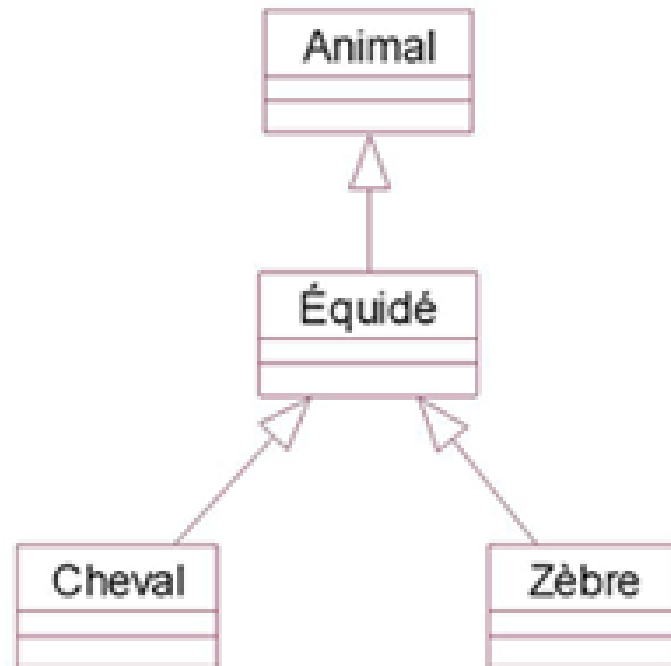
- «**instantiate**» : le client est une fabrique d'objets du fournisseur. Une fabrique est une classe dont le seul but est de créer des instances d'une ou de plusieurs autres classes. La figure 6.38 illustre une fabrique. En effet, la classe Client possède une méthode publique créeFournisseur qui crée une nouvelle instance de la classe Fournisseur.
- «**permit**» : la classe Fournisseur autorise la classe Client à accéder à plusieurs ou à la totalité de ses attributs privés et/ou de ses méthodes privées.
- «**refine**» : la classe Client spécialise la classe Fournisseur. L'utilisation de ce stéréotype à la place de la relation de spécialisation est souvent mise en œuvre lors des premières étapes de la conception des diagrammes UML d'un système, avant que cette relation soit remplacée par la relation de spécialisation.



- Une classe est dite spécifique si toutes ses instances sont également instances d'une autre.
- La classe spécifique est la sous-classe de l'autre.
- La classe générale est la sur-classe.



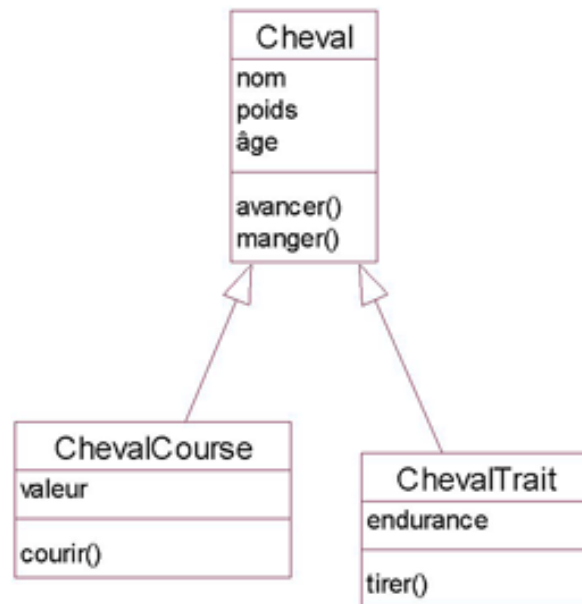
Exemple : *Le cheval est une spécialisation de l'équidé, elle-même spécialisation de l'animal. Le zèbre est une autre spécialisation de l'équidé*



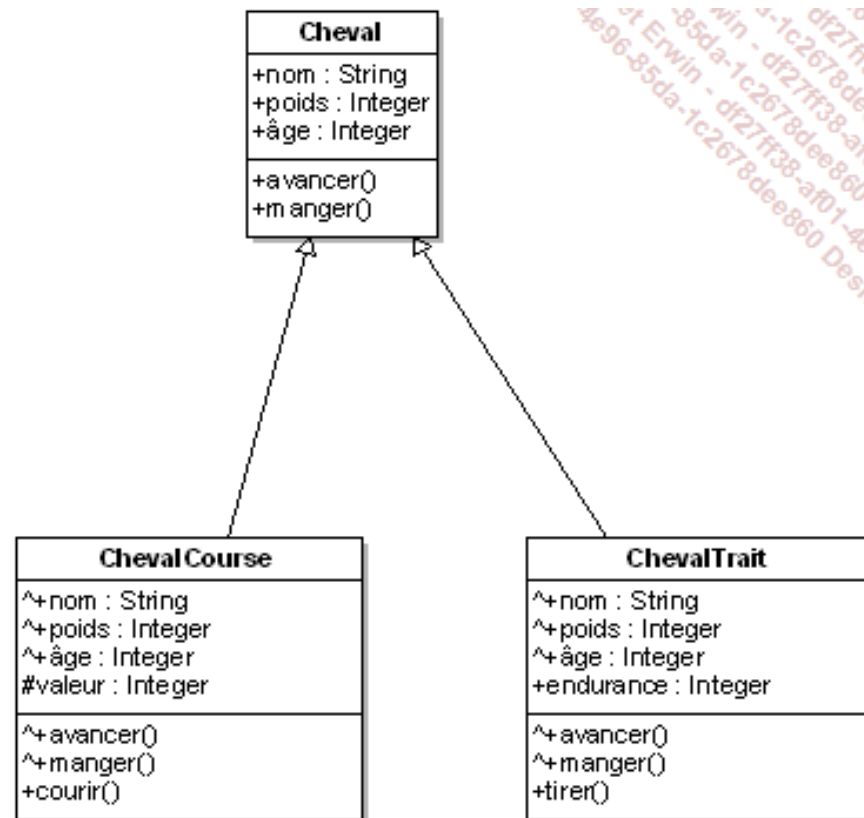
- Les instances d'une classe sont aussi instances de la ou des sous-classes.
- Elles profitent donc en + des attributs/méthodes dans la/les surclasse(s)
- On représente avec un ^
- Si attributs/méthodes privés → hérité mais non visible
- On peut redéfinir méthode/attributs dans la sous-classe (souvent le cas dans les classes abstraites)

*Les attributs et les méthodes de la classe Cheval sont hérités dans ses deux sous-classes.*

*Cet héritage signifie qu'un cheval de course, comme un cheval de trait, possède un nom, un poids, un âge et qu'il peut avancer et manger.*



- Faisons *apparaître la même hiérarchie en indiquant les attributs et les méthodes hérités qui sont alors précédés du symbole ^*. Le type et l'encapsulation des attributs et des méthodes ont également été ajoutés.

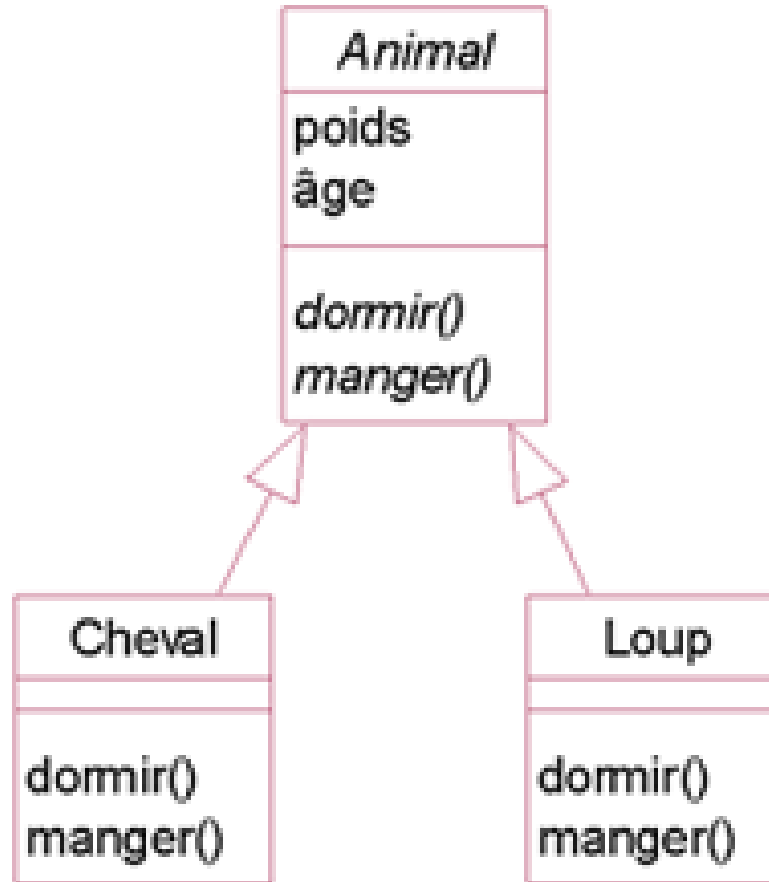




## Classes concrètes ou abstraites

Exemple : *Un animal distinct selon la classe possédant pour la classe Animal, le abstraites.*

*La figure illustre abstraites Animal. rendre concrètes classes Cheval et qu'un loup dort de façon : un loup est herbivore.*



*mais de façon codes de la méthodes*

*sein de la classe ition pour les code) dans les rt debout alors la même l est un*

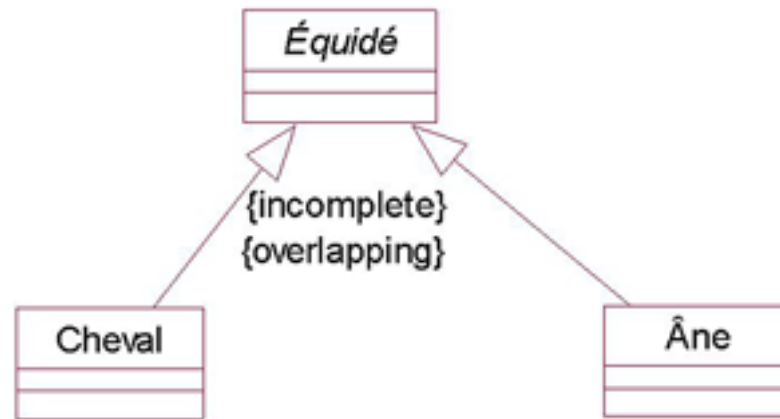
- Deux classes concrètes Cheval et Loup
- Une abstraite *Animal*
- Concrète = possède des instances, modèles complet décrits
- Abstraite = Aucune instance car pas de description complète
  - Vocation de posséder des sous classes
  - Factorise des attributs/méthodes communes
- La factorisation de méthodes communes aux sous-classes se traduit par la seule factorisation de la signature. On parle de méthode abstraite

- En UML : Classe et méthode abstraite sont représenté par le stéréotype « abstract »
- On l'écrit ou on met le nom en italique

- La signature est l'ensemble constitué du nom de la méthode, des paramètres avec leur nom et leur type, ainsi que du type du résultat, à l'exclusion du code de la méthode.
- Toute classe possédant au moins une méthode abstraite est une classe abstraite. En effet, la seule présence d'une méthode incomplète (le code est absent) implique que la classe n'est pas une description complète d'objets.
- Une classe peut être abstraite même si elle n'introduit aucune méthode abstraite.

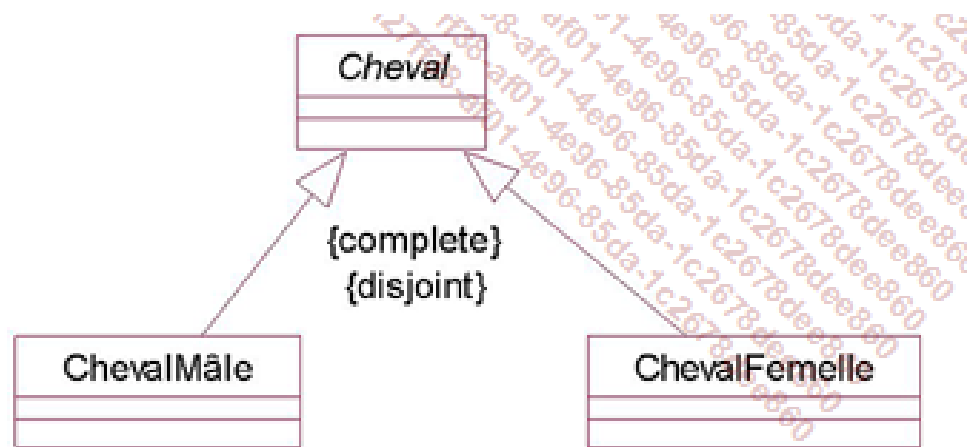
- La contrainte {**incomplete**} signifie que l'ensemble des sous-classes est incomplet et qu'il ne couvre pas la surclasse ou encore que l'ensemble des instances des sous-classes est un sous-ensemble de l'ensemble des instances de la surclasse.
- La contrainte {**complete**} signifie au contraire que l'ensemble des sous-classes est complet et qu'il couvre la surclasse.
- La contrainte {**disjoint**} signifie que les sous-classes n'ont aucune instance en commun.
- La contrainte {**overlapping**} signifie que les sous-classes peuvent avoir une ou plusieurs instances en commun.

Exemple : *une relation d'héritage entre la surclasse Équidé et deux sous-classes : Cheval et Âne. Ces deux sous-classes ne couvrent pas la classe des équidés (d'autres sous-classes existent comme les zèbres). Par ailleurs, il existe les mulets, qui sont issus d'un croisement. Ils sont à la fois des chevaux et des ânes. D'où l'utilisation des contraintes {incomplete} et {overlapping}.*



*Exemple: Une autre relation d'héritage entre la surclasse Cheval et deux sous-classes : ChevalMâle et ChevalFemelle.*

*Ces deux sous-classes couvrent la classe des chevaux et il n'existe aucun cheval qui soit à la fois mâle et femelle. D'où l'utilisation des contraintes {complete} et {disjoint}.*

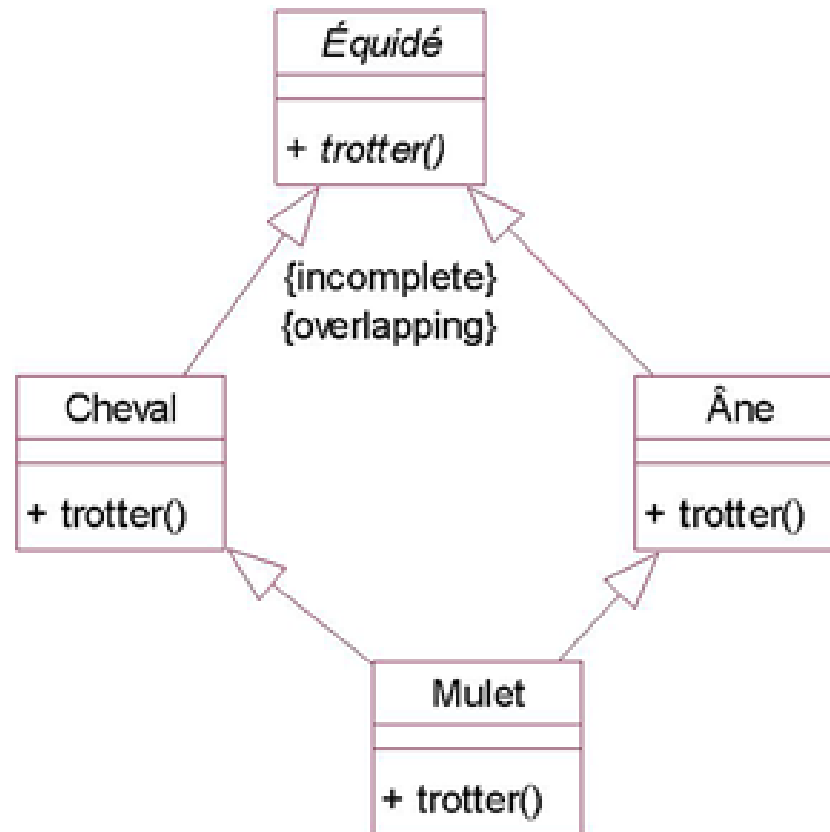


- Une sous-classe hérite de plusieurs surclasses
- Pose un gros problème = une même méthode (dotée de la même signature) est héritée plusieurs fois la sous-classe.
  - Il faudra préciser un critère pour choisir parmi celles héritées
  - Une Solution : redéfinir la méthode dans la sous-classe
- Idem pour les attributs de même : on utilisera pour résoudre {redefines nomAttribut} pour le redef dans la sous-classe



- Si l'utilisation de l'héritage multiple est bien sûr possible en modélisation, il est souvent nécessaire de transformer les diagrammes de classes pour le supprimer lors du passage au développement. En effet, rares sont les langages de programmation supportant cette forme d'héritage. Pour cela, il existe différentes techniques parmi lesquelles la transformation de chaque héritage multiple en une agrégation.

- La figure reprend les deux sous-classes Cheval et Âne et introduit une sous-classe commune, à savoir Mulet.
- Elle illustre également la méthode trotter, héritée dans la classe Equidé à partir des sous-classes Cheval et Âne. Cette méthode fruit de leur héritage multiple, car elle est héritée de leur classe mère, Equidé, et elle est héritée de leur classe grand-mère, Mulet.

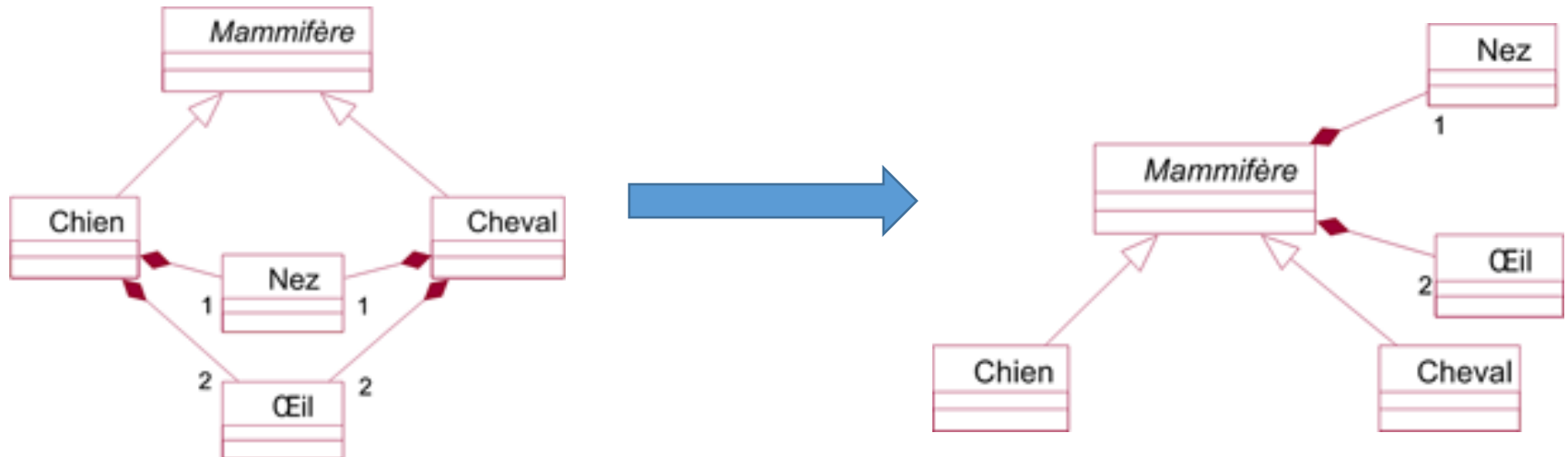


## Factorisation des relations

- Utilisation des classes abstraites pour factoriser

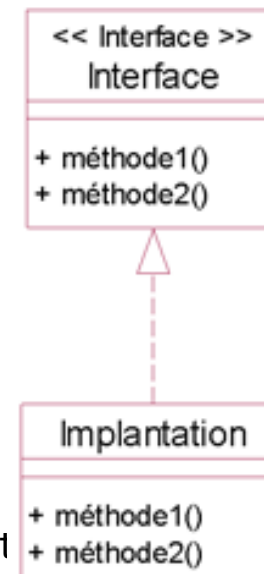
Exemple : Cheval et loup possède deux yeux et un nez (fig1)

*Fig 2 montre qu'une fois créée, la classe abstraite Mammifère surclasse des classes Cheval et Loup, les deux associations de composition sont factorisées.*

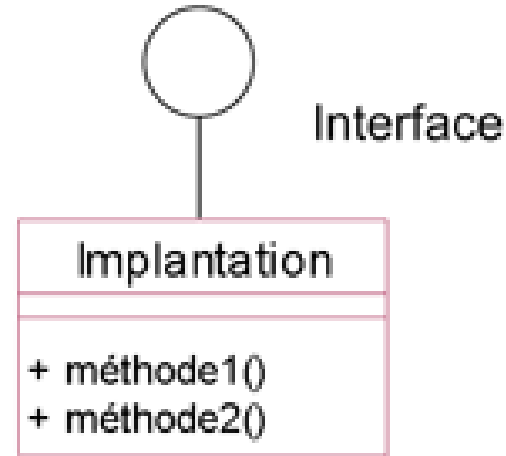


## L'interface

- Est une classe totalement abstraite → 0 attribut avec méthodes abstraites et publiques
- Ne contient aucun élément d'implantation des méthodes
- Représente avec le stéréotype « interface »
- Implémentation des méthodes dans les sous-classes  
→ On parle de relation de réalisation
- Représentée par un trait pointillé



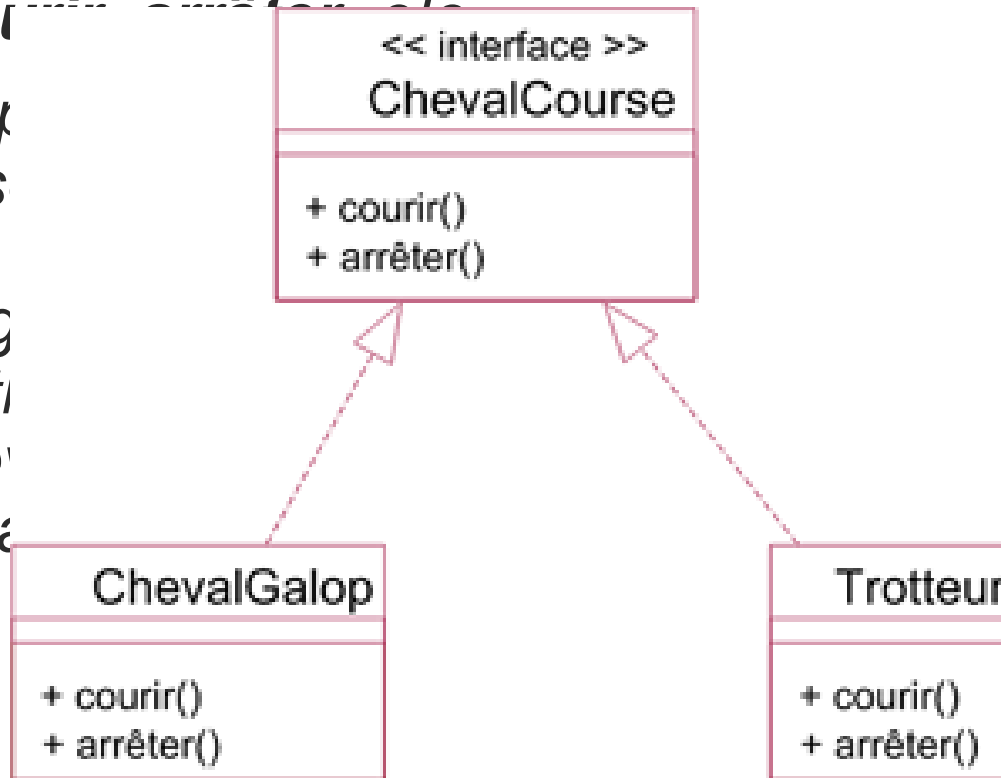
- Représentation en lollipop



- Une même classe peut réaliser plusieurs interfaces. Il s'agit d'un cas particulier de l'héritage multiple. En effet, aucun conflit n'est possible car seules les signatures des méthodes sont héritées dans la classe de réalisation. Si plusieurs interfaces contiennent la même signature, cette signature est implantée par une seule méthode dans la classe commune de réalisation.

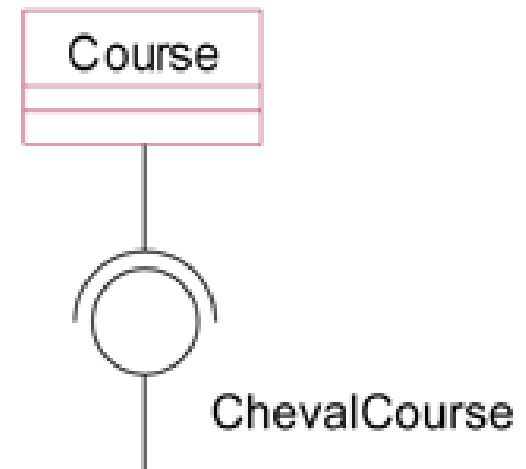
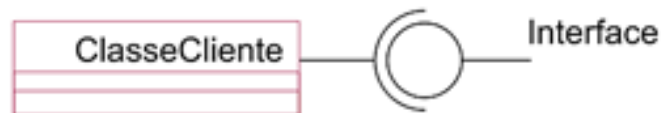
Exemple : *Un cheval de course peut être considéré comme une interface. Celle-ci est composée de plusieurs méthodes : courir, arrêter, etc.*

*L'implantation du galop ou du trot se fait selon l'une ou l'autre des méthodes courir ou arrêter. Dans ce cas, courir signifie galoper et arrêter signifie trotter. L'interface **ChevalCourse** est donc une interface différente due à la nature des méthodes.*



*galop ou  
trot pour  
le galop,  
le trotteur),  
l'implantation*

- Classe qui dépend d'une interface pour ses opérations
  - On utilise l'interface comme type dans la classe
  - On dit que la classe est cliente de l'interface



Exemple : *Une course a besoin de chevaux de course pour être organisée. La classe Course dépend donc de l'interface ChevalCourse*

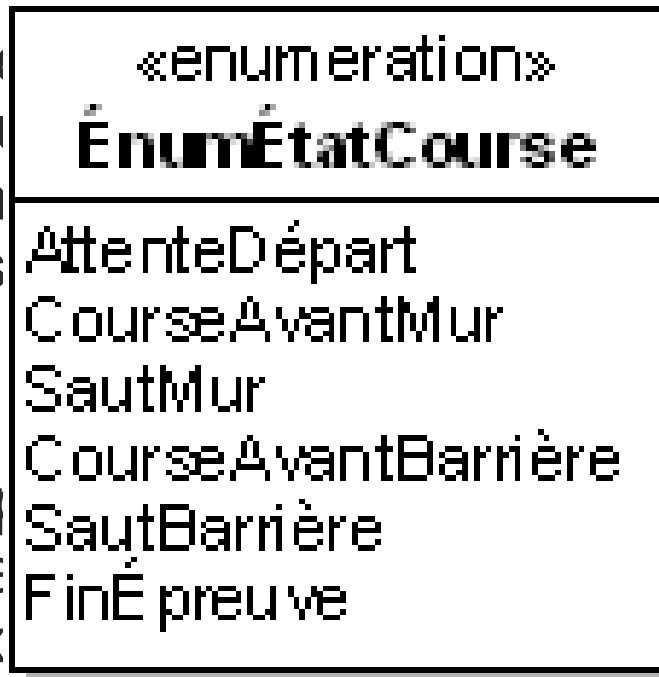




## Les stéréotypes de classe

- «**abstract**» : ce stéréotype indique qu'une classe est abstraite.
- «**auxiliary**» : il s'agit d'une classe secondaire d'un système. Les classes secondaires servent au support des classes principales.
- «**focus**» : il s'agit d'une classe principale d'un système. Ce stéréotype s'oppose au stéréotype «**auxiliary**».
- «**implementation class**» : une classe d'implantation fournit la réalisation d'une classe «**type**».
- «**type**» : une classe «**type**» correspond à un type de données abstrait en introduisant les attributs et les méthodes abstraites, mais sans fournir aucune réalisation. Une telle classe est abstraite et se distingue d'une interface par l'introduction d'attributs.

- **«interface»** : ce stéréotype précise qu'une classe est une interface.
- **«utility»** : ce stéréotype indique une classe utilitaire qui ne possède qu'une seule propriété {readOnly} et aucune méthode de classe. Elle n'est pas instanciée.
- **«enumeration»** : ce stéréotype indique une classe particulière qui introduit un type énuméré. Ses valeurs sont précisées par une liste de littéraux. Ces valeurs sont appelées littéraux.



de classe utilitaire  
e dotés de la  
stantes ainsi que des  
pas vocation à être

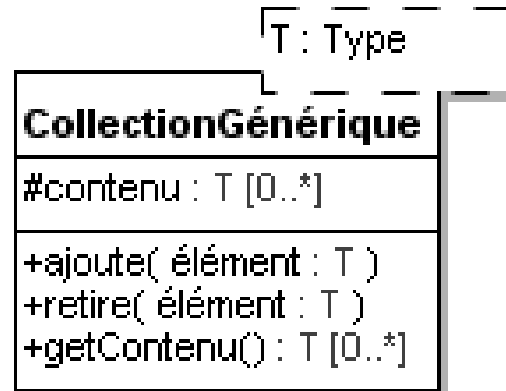
e classe particulière  
valeurs sont  
Ces valeurs sont

Une énumération est représentée graphiquement par cette liste de littéraux.

## Les classes templates

- Modèle de classe
- Possède paramètres qui donne aspect générique
- On type les paramètres
  - En UML , type **Type** (même niveau que Int, String...)
- Valeur fixé par liaison (binding) entre classe template et instancié
- Equivalent à la relation objet – classe
- Syntaxe : nomParamètre : **type**=valeurDéfaut  
Type et valeur par défaut son optionnel

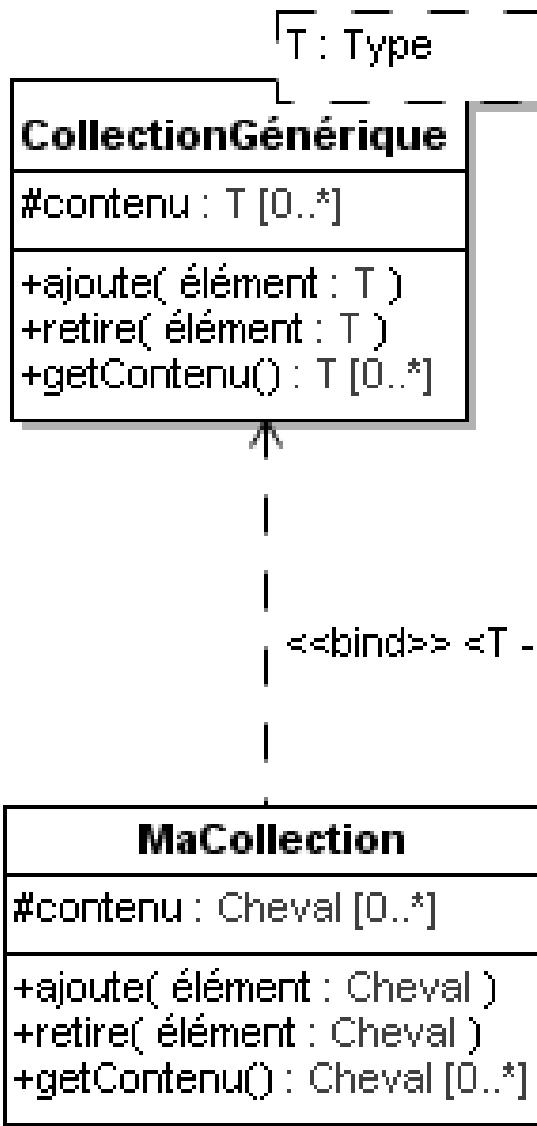
Exemple : Voyons la classe template **CollectionGénérique**. Elle possède un paramètre **T** ayant pour type la classe **Type**. Elle implante une collection où les éléments sont stockés dans un tableau introduit par l'attribut contenu. La méthode **ajoute** insère un élément, la méthode **retire** enlève un élément et la méthode **getContenu** retourne son contenu sous la forme d'un tableau.



- La relation de liaison de fixer la valeur de

- `<<bind>>` `<nomPara`

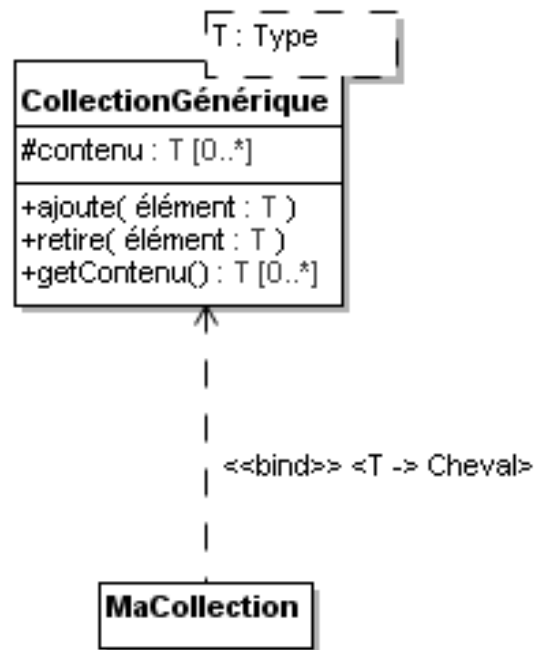
Exemple : *liaison entre collection de chevaux template **CollectionG** à Cheval.*



→ Elle permet

`<<bind>> <T -> Cheval>` *qui décrit ma est fixée*

Exemple : Représentation où la classe **MaCollection** n'est pas détaillée( pas obligatoire)



## Objets ou Instances ?

- Le diagramme de classes est une représentation statique du système. Il peut également montrer les objets, c'est-à-dire, à un moment donné, les instances créées et leurs liens lorsque le système est actif.
- Chaque instance est représentée dans un rectangle qui contient son nom en style souligné et, éventuellement, la valeur d'un ou de plusieurs attributs. (le nom de celle-ci est optionnel)

```
nomInstance : nomClasse
```

- Un attribut

```
nomAttribut = valeurAttribut
```

## Objets ou Instances ?

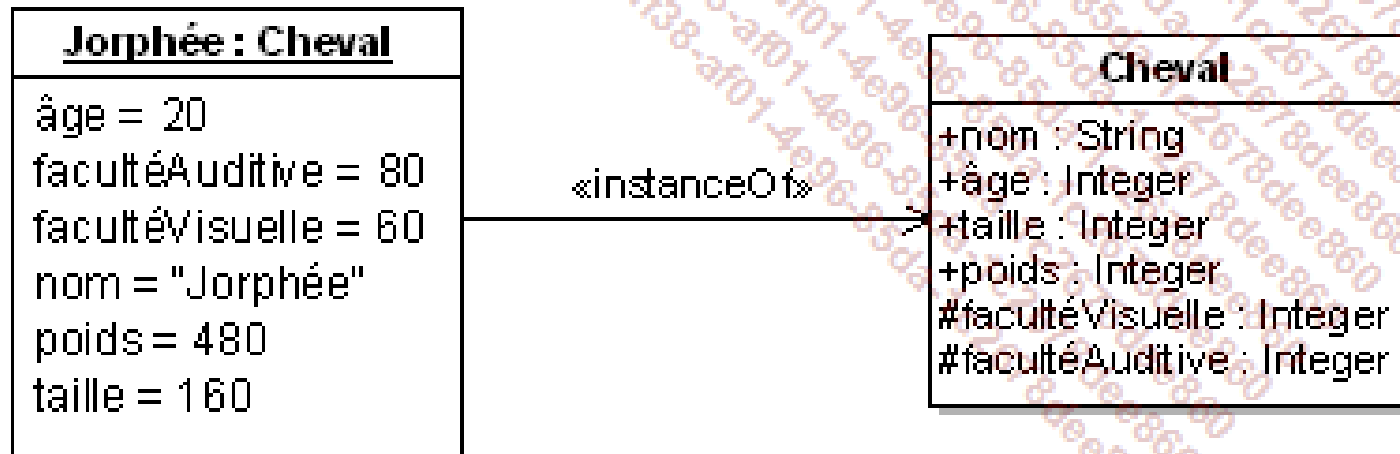
Exemple : Jorphée, une instance de la classe Cheval

<u>Jorphée : Cheval</u>
âge = 20
facultéAuditive = 80
facultéVisuelle = 60
nom = "Jorphée"
poids = 480
taille = 160



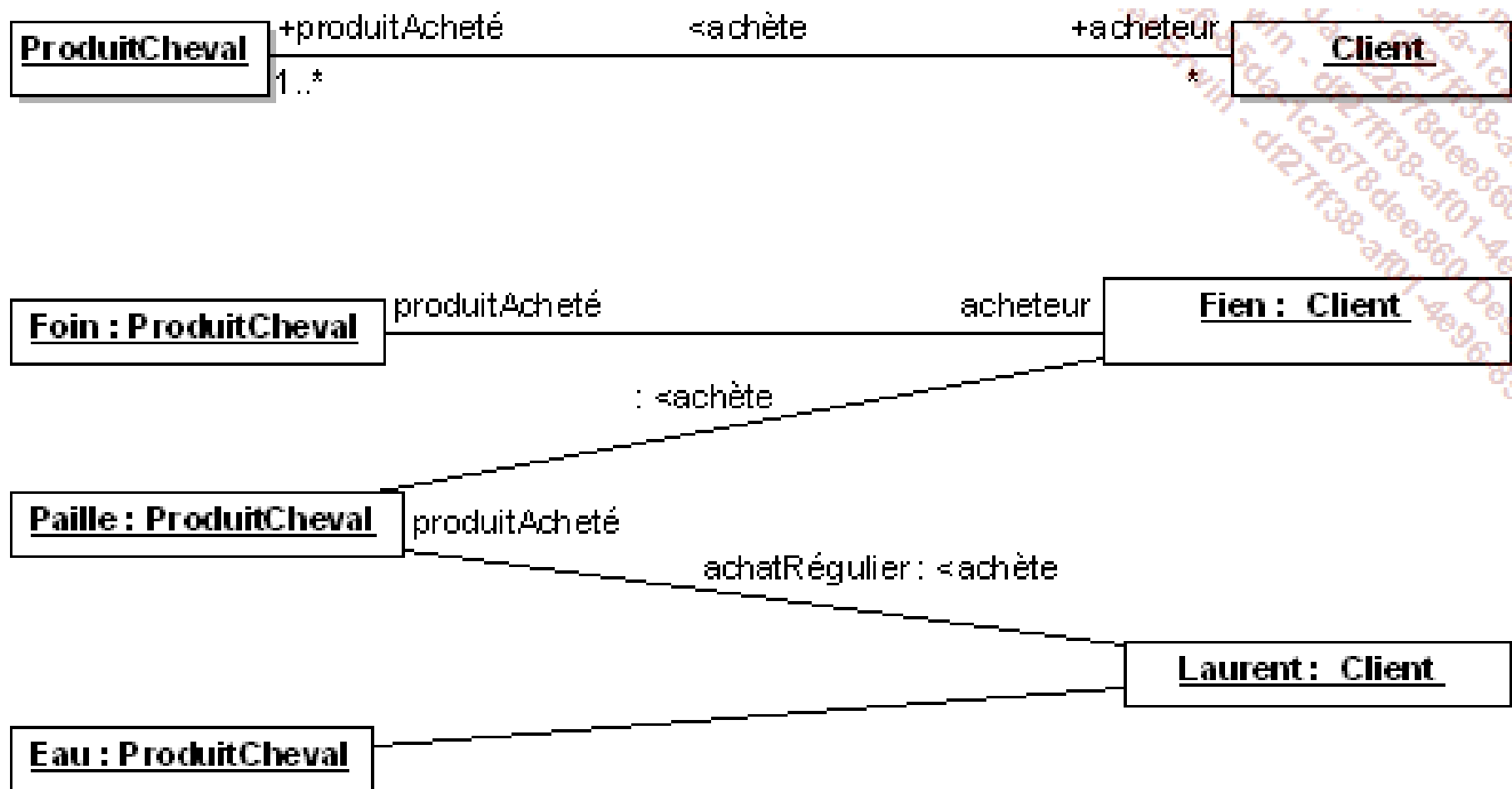
## Objets ou Instances ?

- La relation d'instanciation :
  - Décrit le lien entre instance et sa classe
  - Décrit dans le nom de l'instance suffixé par la classe
  - Peut aussi par relation de dépendance muni du stéréotype « instanceOf » (moins courant)



## Objets ou Instances ?

- Les liens entre instances = des simples traits continus
- On dit qu'ils sont les occurrences de relations interobjets
- On les souligne comme pour les objets et les suffixe comme pour l'association
- On peut ajouter le nom des rôles



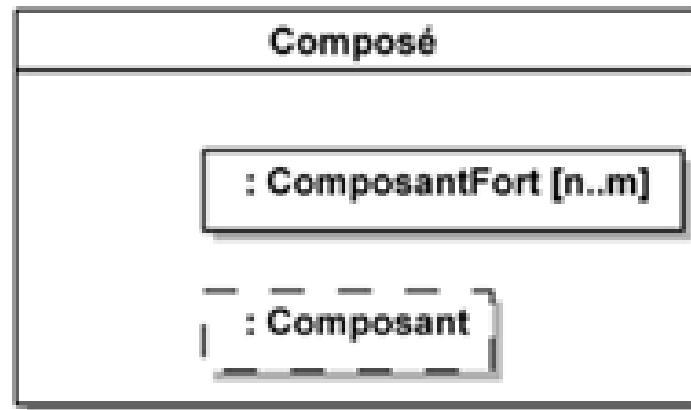


- Description d'un objet composé
- Complète le diagramme de classe
- L'objet composé = classificateur
- Les composants = partie
- Association à une classe (non décrite en détail)



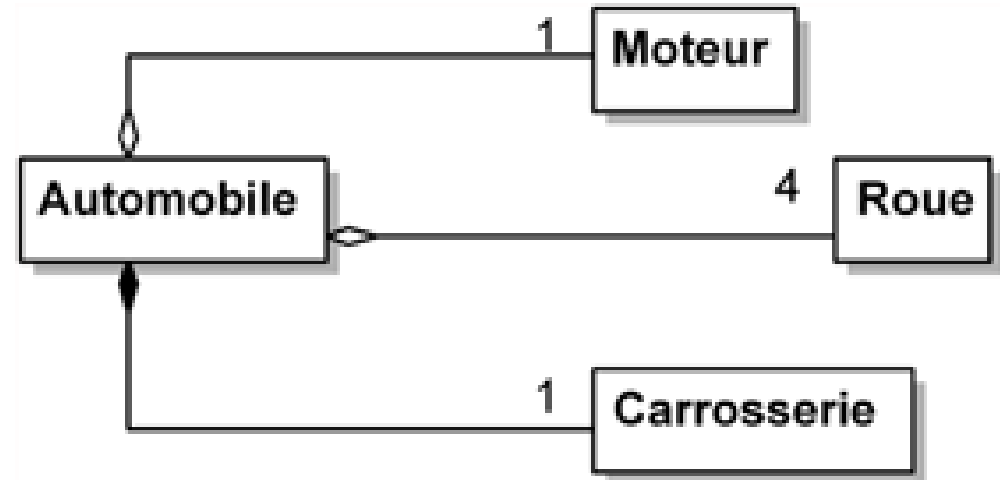
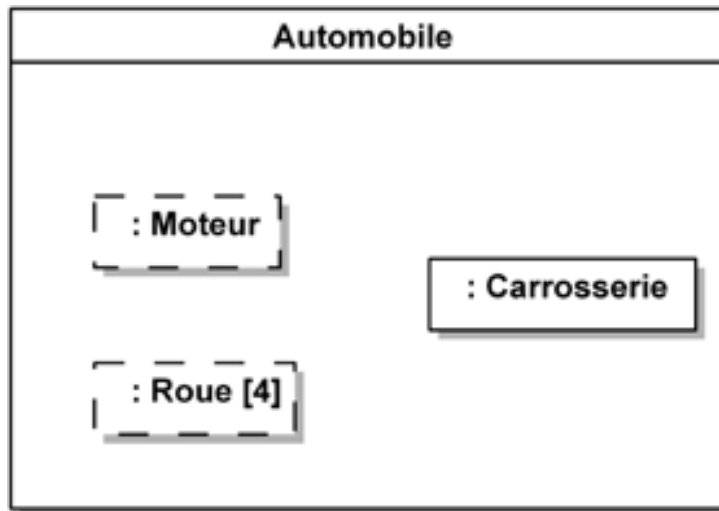
- Un composant venant d'une composition forte et un d'une agrégation

- La figure montre le diagramme de structure composite correspondant à cet objet. Les composants sont intégrés au sein du classificateur qui décrit l'objet composé. Les parties possèdent un type qui est la classe du composant. La cardinalité est indiquée entre crochets. Par défaut, elle vaut 1. Un composant issu d'une agrégation est représenté par une ligne en pointillés, un composant issu d'une composition forte est représenté par une ligne continue.



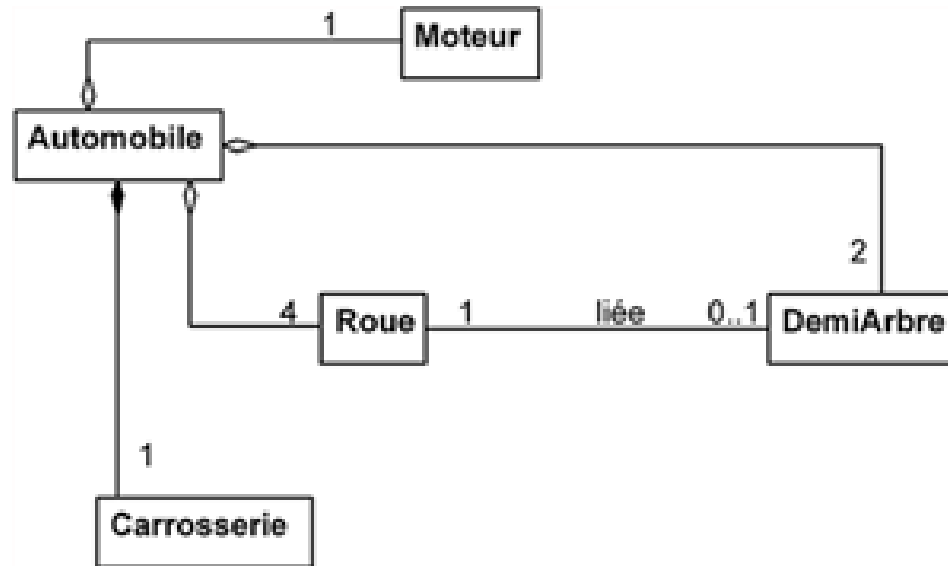
# Le diagramme de structure composite

Exemple : Une voiture



Attention : Dans le diagramme de structure composite, Moteur, Carrosserie et Roue ne sont pas des classes mais des parties. Une partie est toujours prise en compte au sein d'un classificateur

- Ajoutons le demi arbre pour les roues avant (une association)

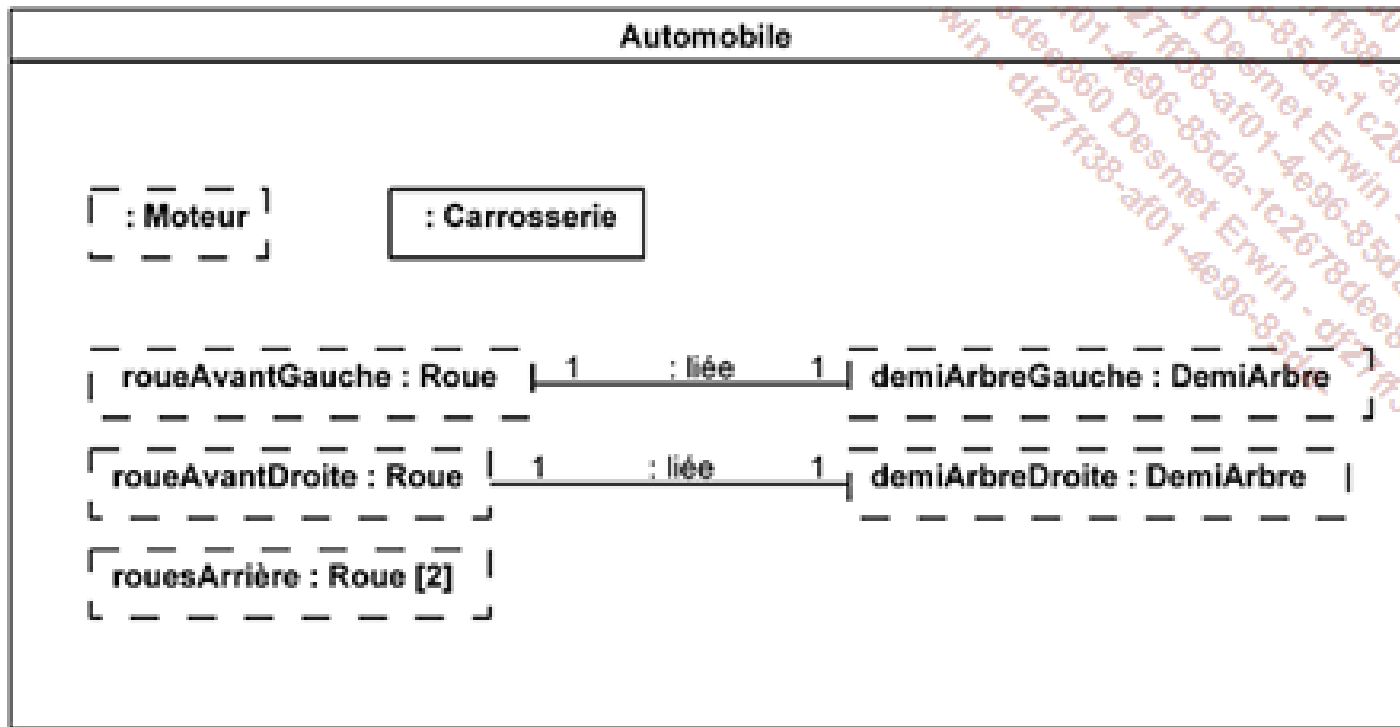


- On ne sait pas dans ce diagramme précisé qu'on parle des roues avant ou arrières sans incorporé deux sous-classes.
- On évitera de le faire pour ne pas surchargé.
- Attention à la cardinalité donc



# Le diagramme de structure composite

- Dans un diagramme de structure composite c'est ok
- Le rôle décrit l'utilisation du composant
- Notation est différente du diagramme d'objets

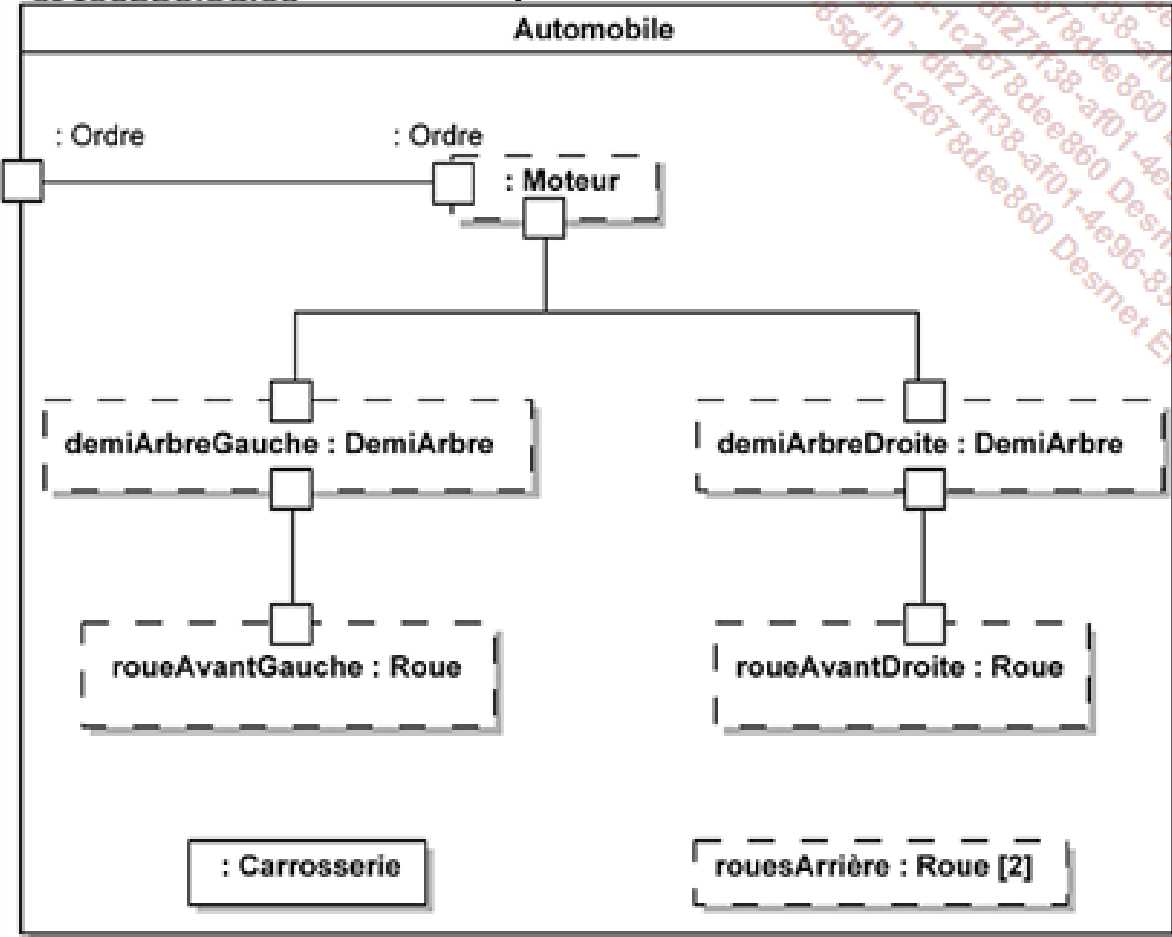


- Les connecteurs peuvent également relier des parties entre elles au travers de ports. Un port est un point d'interaction. Il possède une interface qui constitue son type et définit l'ensemble des interactions possibles. Les interactions conduites par un port se font avec les autres ports qui lui sont liés par un connecteur.
- Un port peut également être introduit au niveau du classificateur. Un tel port a alors pour objectif de servir de passerelle entre les parties internes du classificateur et les objets externes à celui-ci.
- Du point de vue de l'encapsulation, un tel port est généralement public. Il est alors connu en dehors du classificateur.

- Un port défini au niveau du classificateur peut aussi être défini comme privé. Il est alors réservé à une communication interne entre le classificateur et ses parties. Il ne fait pas office de passerelle entre l'intérieur et l'extérieur du classificateur.
- Une partie peut posséder plusieurs ports, chacun possédant sa propre interface. Plusieurs ports d'une même partie peuvent être typés avec la même interface. Il est alors possible de les distinguer en leur affectant des noms de rôle différents, à l'instar de ce qui se fait pour les parties et les connecteurs.

# Le diagramme de structure composite

- Ajoutons
- Entre mo
- Connecti
- Ajoutons

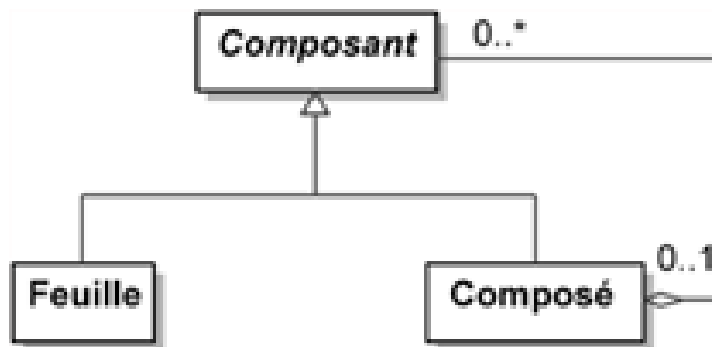


ent

- Au niveau des interactions, cette figure illustre que :
  - La classe Automobile peut interagir avec l'extérieur pour recevoir des ordres destinés à son moteur et qui lui sont transmis.
  - Le moteur communique avec les demi-arbres (transmission du mouvement).
  - Chaque demi-arbre communique avec les roues (transmission du mouvement).

- La collaboration :
  - Interagir ensemble pour une fonction du système
  - Chaque objet à sa fonction
  - Nomenclature ressemble à celle de la structure
  - Décrit souvent des patterns de conception

Exemple : pattern composite

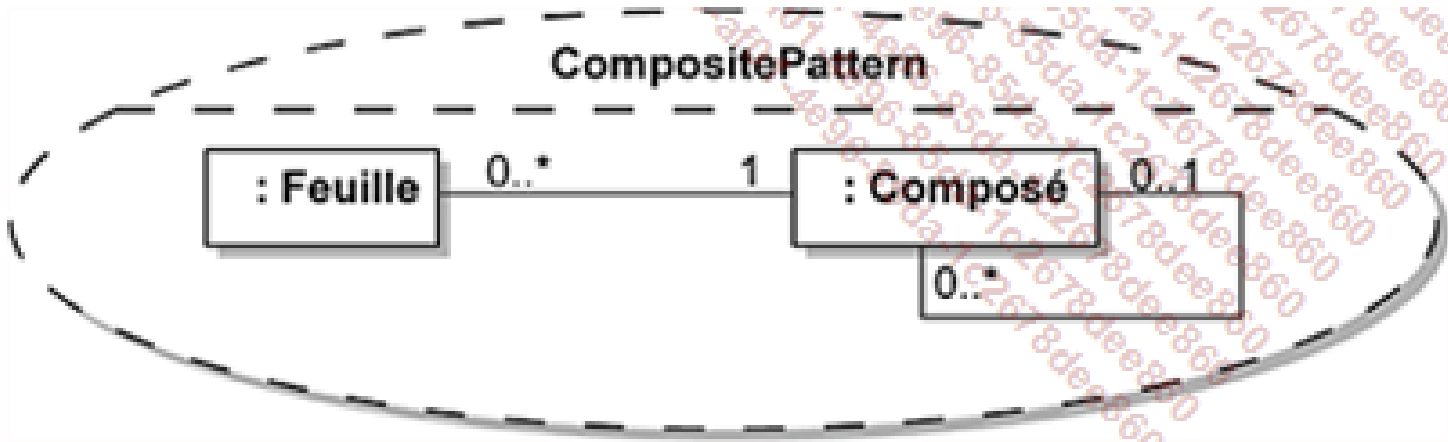


## Le diagramme de structure composite

- Le but du pattern Composite est d'offrir un cadre de conception d'une composition d'objets dont la profondeur est variable. Un composant est soit une feuille soit un objet composé, à son tour, de feuilles et d'autres composés. L'un des exemples qui illustrent le mieux ce pattern est celui du système de fichiers du disque dur d'un ordinateur personnel. Il est composé de fichiers et de répertoires. Chaque répertoire peut, à son tour, contenir des fichiers ou d'autres répertoires.
- Une collaboration décrivant le pattern Composite est illustrée à la figure suivante. Celle-ci présente les deux parties de la collaboration, à savoir celle qui a pour classe Feuille et celle qui a pour classe Composé. Toute feuille est nécessairement contenue dans un et un seul composé, c'est-à-dire qu'il existe au moins un composé. Un composé peut être contenu dans un autre composé ou non (cas du composé racine).

## Le diagramme de structure composite

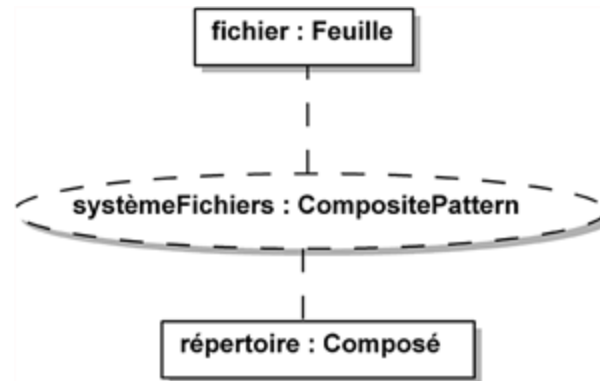
Une collaboration ne remplace pas un diagramme de classes mais le complète. C'est ce que nous avons dit au début : le diagramme de structure composite n'a pas vocation à remplacer le diagramme de classes mais à le compléter.





## Le diagramme de structure composite

- Le diagramme de structure composite offre la possibilité de décrire une application d'une collaboration en fixant les rôles des parties, ce que nous avons fait à la figure suivante en prenant notre exemple de système de fichiers comme application de la collaboration du pattern Composite.



- 1. La hiérarchie des chevaux

Soit les classes Jument, Étalon, Poulain, Pouliche, Cheval, ChevalMâle et ChevalFemelle ainsi que les associations père et mère.

Établissez la hiérarchie des classes en y faisant figurer les deux associations.

Utilisez les contraintes {incomplete}, {complete}, {disjoint} et {overlapping}.

Introduisez la classe Troupeau. Établissez l'association de composition entre cette classe et les classes déjà introduites.

## ● 2. Les produits pour chevaux

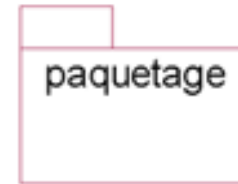
Modélisez les aspects statiques du texte suivant sous la forme d'un diagramme de classes.

Une centrale des chevaux vend différents types de produits pour chevaux : produits d'entretien, nourriture, équipement (pour monter le cheval), ferrures.

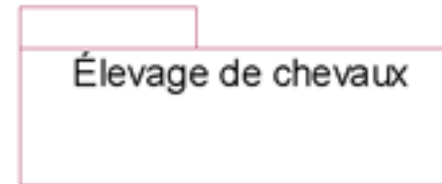
Une commande contient un ensemble de produits avec, pour chacun d'eux, la quantité. Un devis est éventuellement établi avant le passage de la commande. En cas de rupture de stock, la commande peut engendrer plusieurs livraisons si le client le désire. Chaque livraison donne lieu à une facture.

# Quizz

- En Uml, un paquetage = un dossier

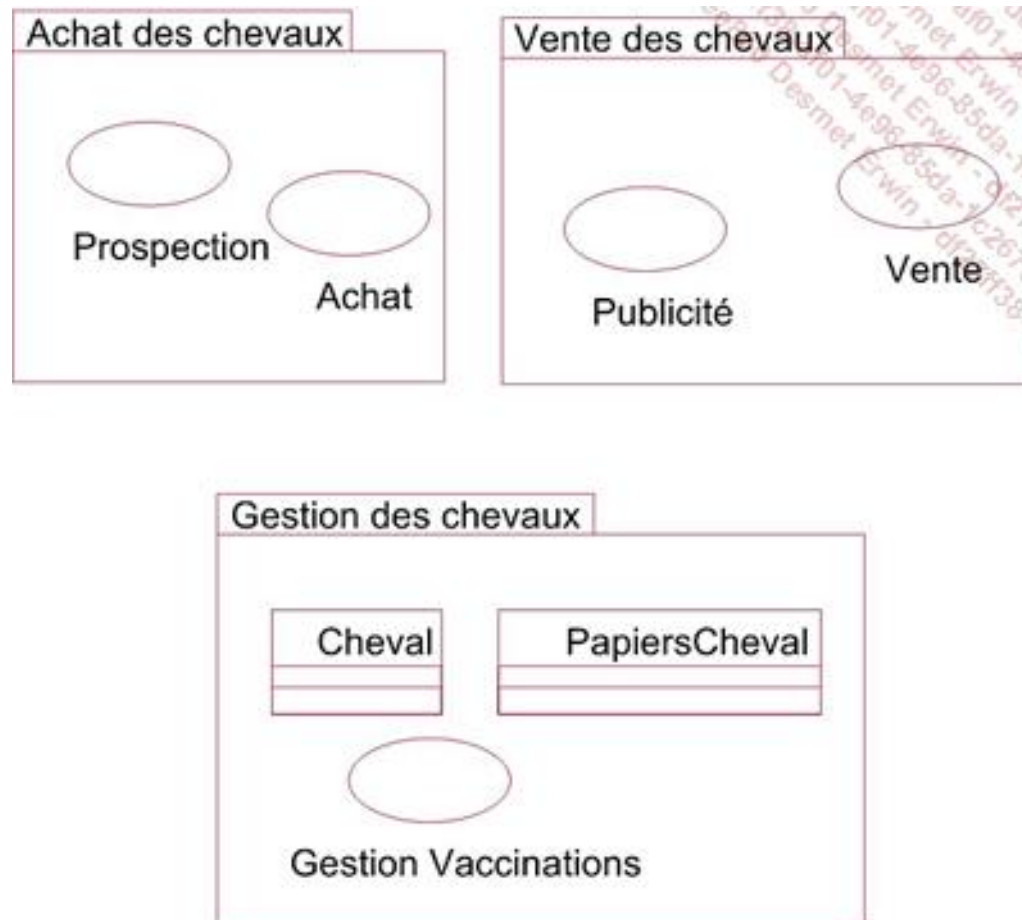


Exemple avec nos chevaux :

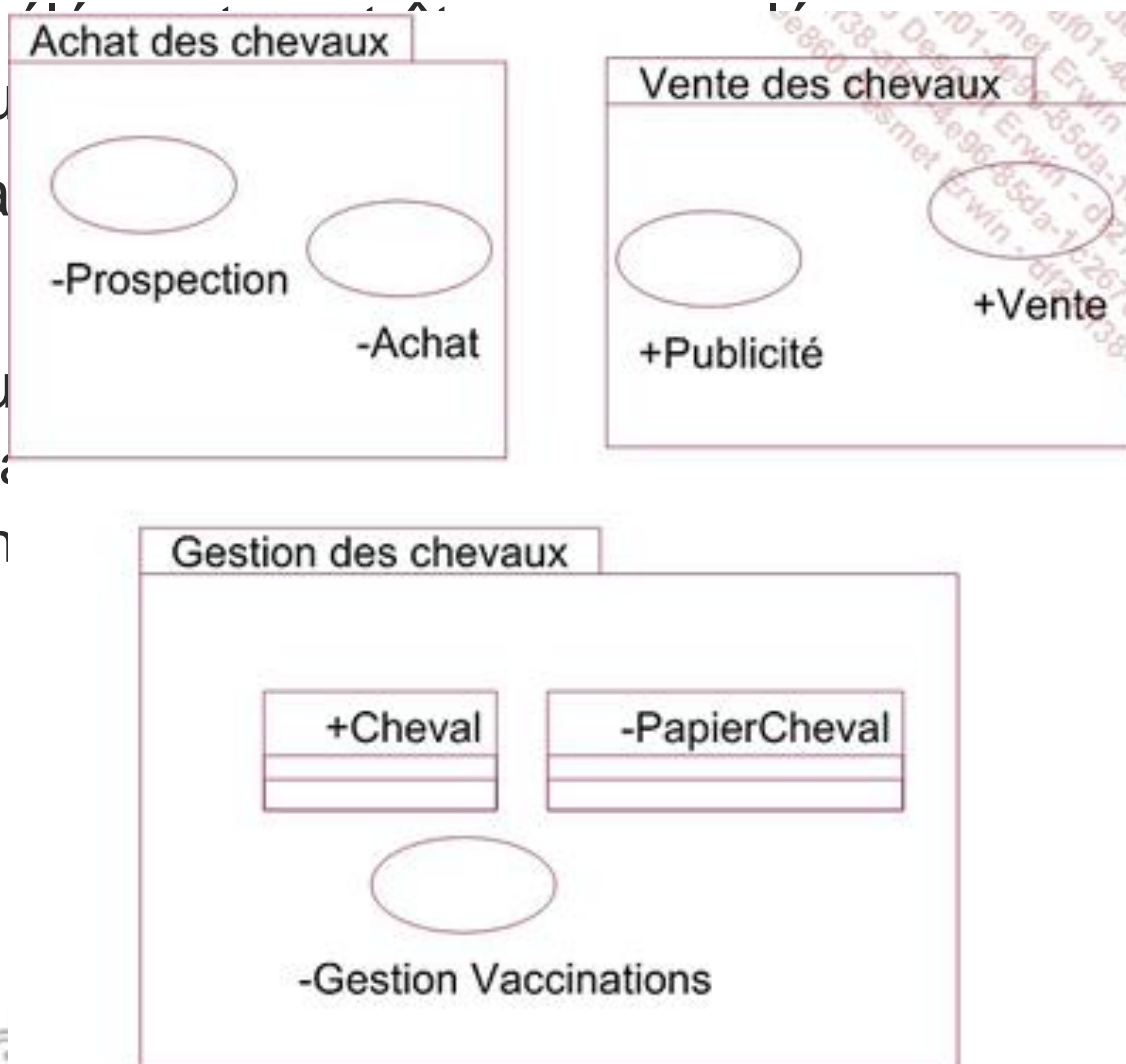


- Le contenu est décrit par un diagramme de paquetage
  - Regroupe les éléments avec leurs représentations graphiques
  - Peuvent être : classes, composants, cas d'utilisation, paquetages,...

Exemple : *Le contenu du paquetage Élevage de chevaux est illustré par son diagramme. Celui-ci contient trois paquetages qui contiennent des cas d'utilisation et des classes*



- Chaque élément est encapsulé dans un rectangle
- L'extérieur du rectangle est réservé à la description globale de l'élément
- Par défaut, les éléments sont classés en trois catégories :
  - + = processus
  - - = entité

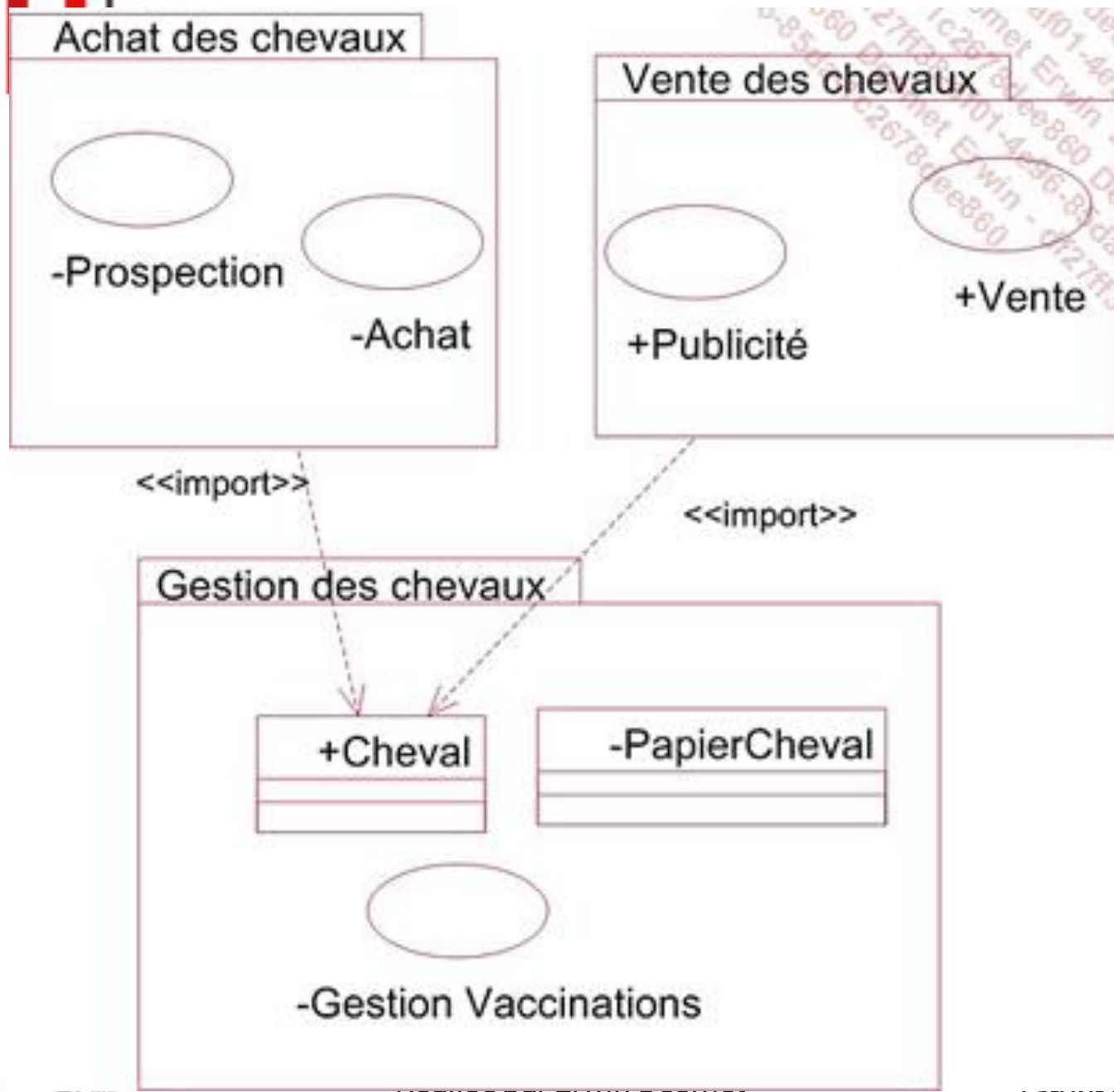


- Relations entre les paquetages :
  - **La relation d'importation** consiste à amener dans le paquetage de destination un élément du paquetage d'origine. L'élément fait alors partie des éléments visibles du paquetage de destination.
  - **La relation d'accès** consiste à accéder, depuis le paquetage de destination, à un élément du paquetage d'origine. L'élément ne fait alors pas partie des éléments visibles du paquetage de destination.
- Possible seulement si l'élément est visible dans le paquetage d'origine



- Peut s'appliquer à un paquetage complet (tous éléments visible du paquetage d'origine)
- On les dit de dépendances et se schématise avec les stéréotype : **import** et **access**

*Exemple : Dans le paquetage Élevage de chevaux, les paquetages Achat de chevaux et Vente de chevaux importent la classe Cheval. Le résultat aurait été le même si ces deux paquetages avaient importé le paquetage Gestion des chevaux car la classe Cheval est la seule à être publique.*



- La relation de fusion entre deux paquetages
  - Engendre la fusion du contenu non privé de la destination vers le paquetage de source.



- Règles générales:

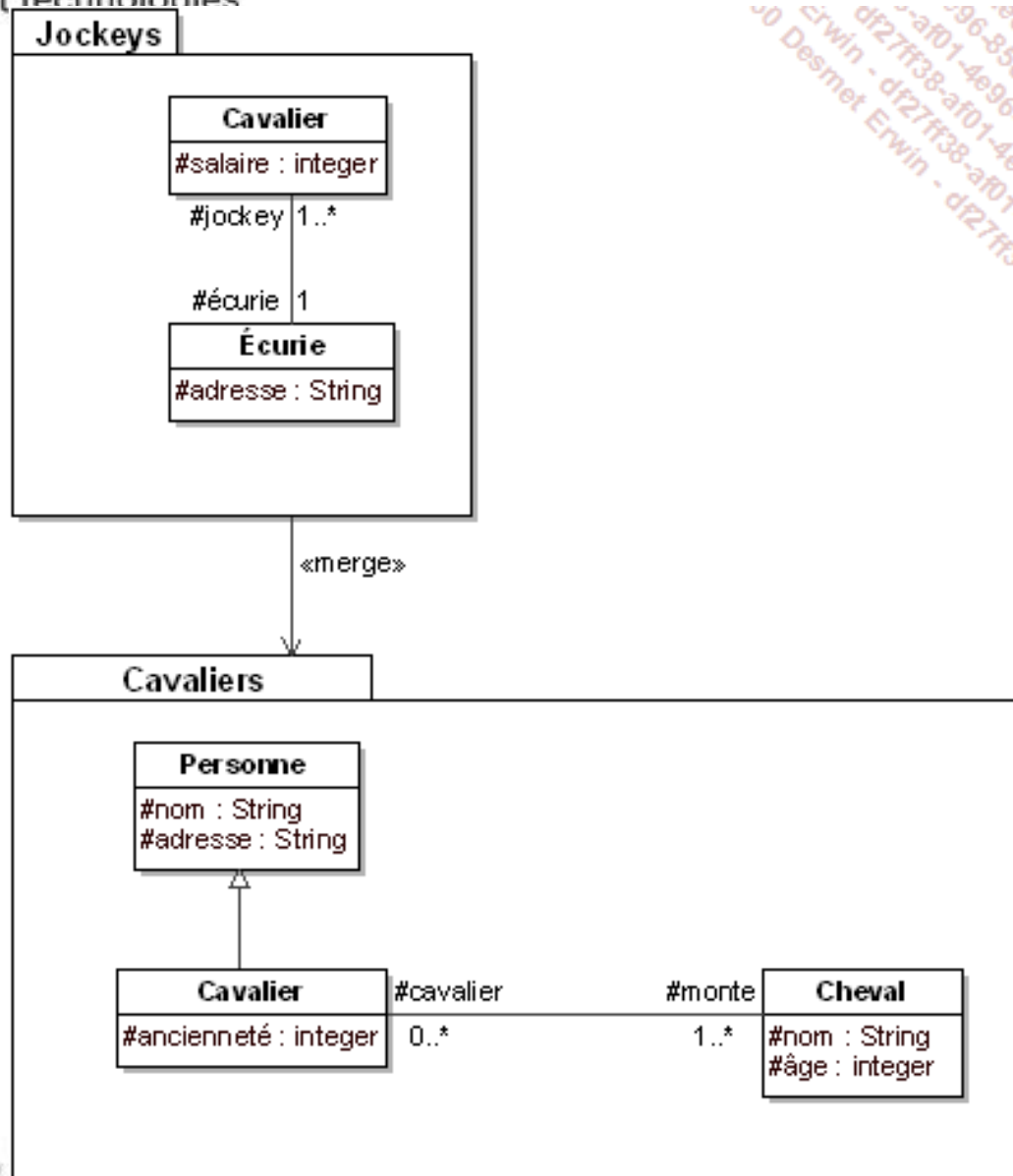
- La relation de fusion est transformée en une relation d'importation en conservant la même orientation. Les éléments publics du paquetage de destination sont importés dans le paquetage d'origine.
- Pour chaque élément du paquetage de destination qui peut être spécialisé (comme les classes, les cas d'utilisation, etc.) et pour lequel il n'existe pas un élément de même nom dans le paquetage d'origine, un nouvel élément de même nom est créé dans ce paquetage. Ensuite, une relation de spécialisation est introduite depuis tous les éléments du paquetage d'origine vers les éléments du paquetage de destination qui possèdent le même nom. Toutes les propriétés ainsi héritées dans les éléments du paquetage d'origine sont redéfinies. Ces propriétés redéfinies appartiennent par conséquent aux éléments du paquetage d'origine.

- Les relations de spécialisation existantes entre les éléments du paquetage de destination sont réintroduites entre les éléments de même nom dans le paquetage d'origine.
- Pour chaque paquetage présent dans le paquetage de destination et pour lequel il n'existe pas un paquetage de même nom dans le paquetage d'origine, un nouveau paquetage de même nom est créé dans ce paquetage. Ensuite, une relation de fusion est définie entre tous les paquetages du paquetage d'origine et les paquetages du paquetage de destination qui possèdent le même nom. Les règles de la fusion sont ainsi appliquées de façon récursive aux paquetages imbriqués.

- Les relations d'importation et d'accès existantes entre les paquetages du paquetage de destination sont réintroduites entre les paquetages de même nom dans le paquetage d'origine.
- Les éléments du paquetage de destination qui ne peuvent être généralisés ou spécialisés sont copiés dans le paquetage d'origine. Leurs relations avec les autres éléments dans le paquetage d'origine deviennent identiques à celles du paquetage de destination.
- Attention : La fusion entre deux paquetages est introduite avec un statut de relation, et non un statut d'opération. L'application des règles de fusion détermine la description du paquetage d'origine.

Exemple : *Le paquetage **Jockeys** est le paquetage d'origine de la relation de fusion. Il introduit des descriptions spécifiques aux cavaliers professionnels (salaire, écurie) alors que le paquetage de destination, le paquetage **Cavaliers**, introduit une description générale des cavaliers. Tous les éléments du paquetage **Cavaliers** sont publics.*

# La structuration des éléments de modélisation



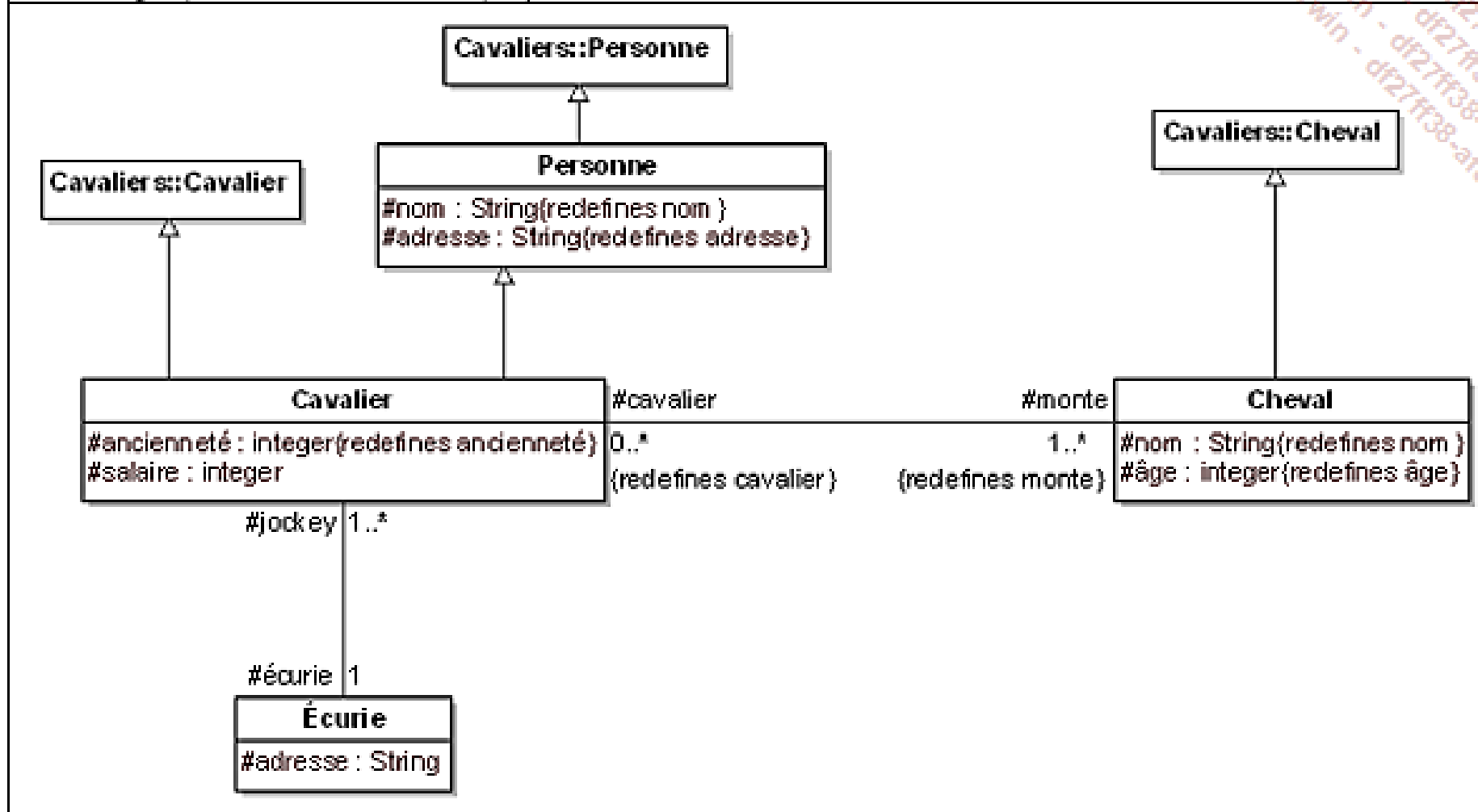


*La figure suivante illustre la description du paquetage **Jockeys** telle qu'elle est déterminée par l'application des règles de la relation de fusion.*

*Les classes dont le nom est préfixé par **Cavaliers::** sont les classes du package **Cavaliers** qui sont importées dans le paquetage **Jockeys** (voir règle 1).*

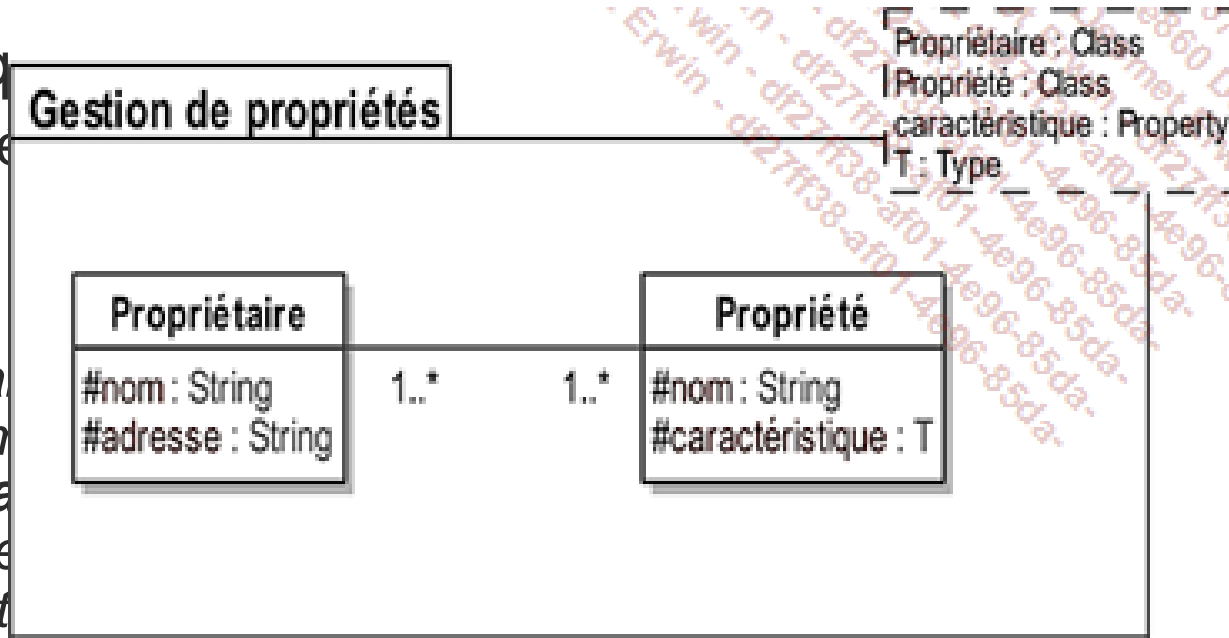
*Les classes **Cheval** et **Personne** ont été introduites dans le paquetage **Jockeys**. Elles héritent des classes **Cheval** et **Personne** du paquetage **Cavaliers** dont elles redéfinissent les attributs hérités (voir règle 2). La classe **Cavalier** qui existait dans le paquetage **Jockeys** hérite maintenant de la classe **Personne** du même paquetage (voir règle 3) et de la classe **Cavalier** du paquetage **Cavaliers** dont elle redéfinit les attributs **ancienneté** et **monte**.*

### Jockeys (résultat de la fusion)



- Les paquets
- Génération

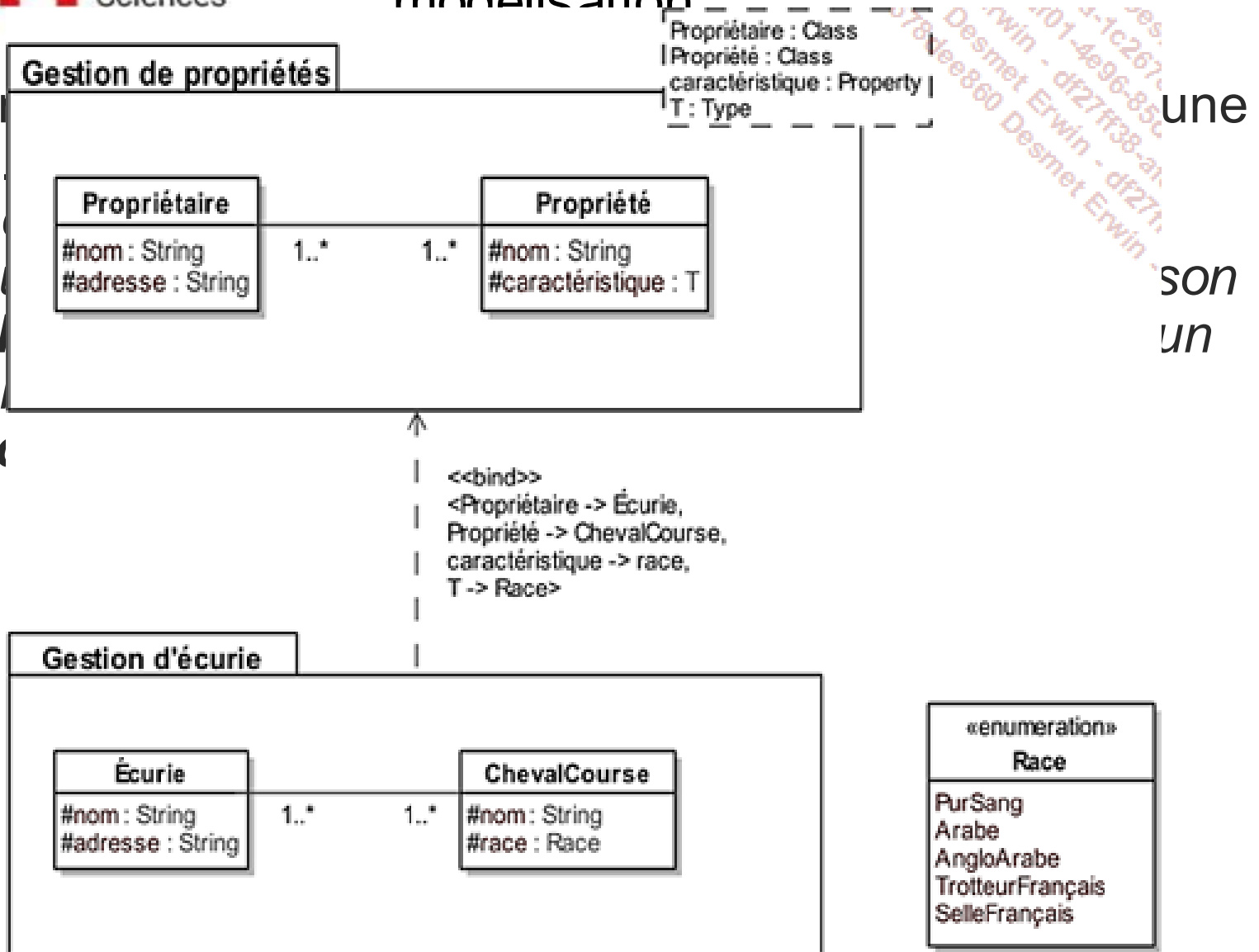
*Exemple*  
l'attribut **ca**  
classe du n  
L'attribut **ca**  
attributs. Le  
classes et t



ie,  
**Class**, la  
crit les  
es les

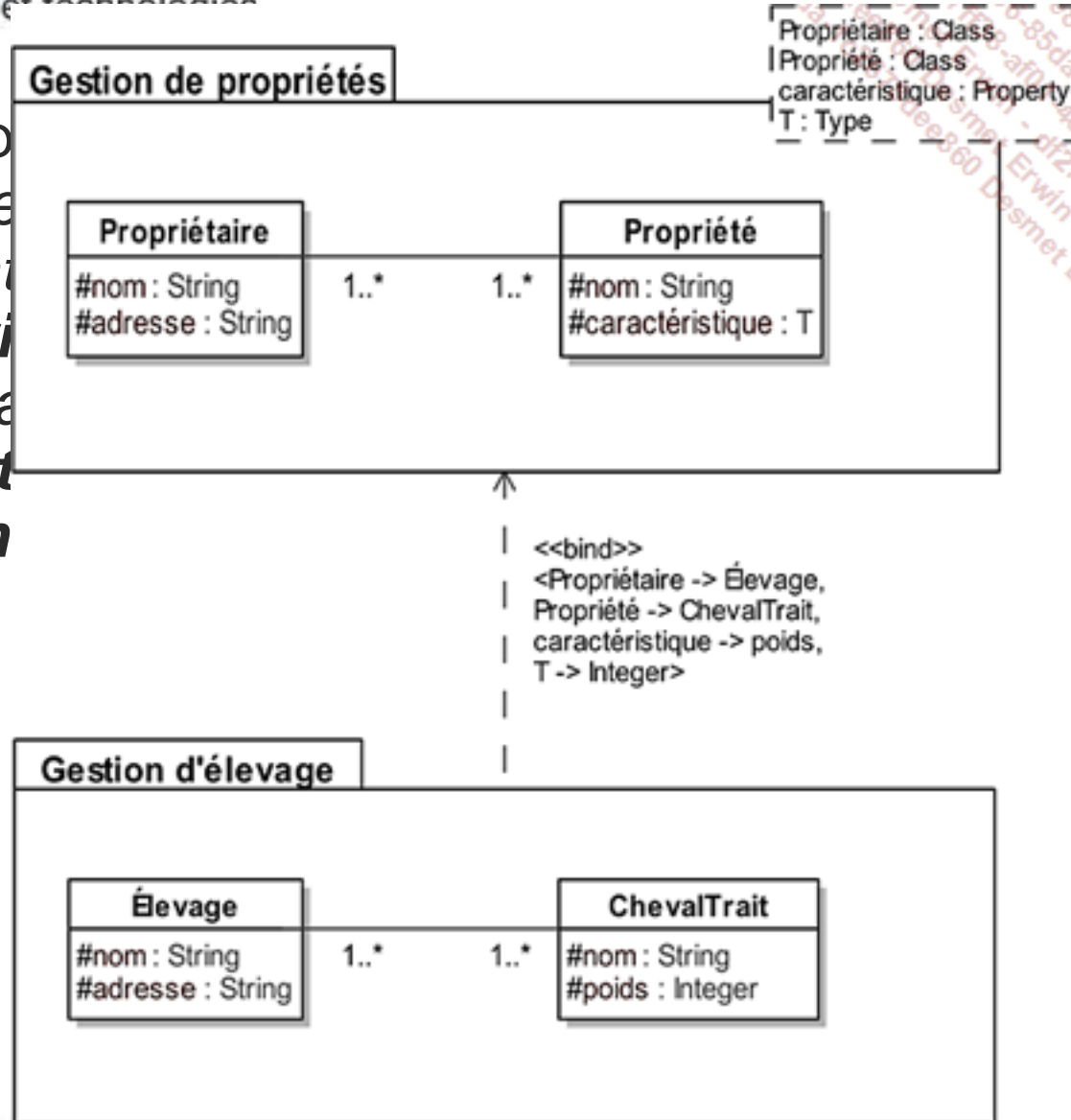
*Ce paquetage décrit de façon générique l'association entre un ou plusieurs propriétaires et leur(s) propriété(s), les indivisions étant prises en compte. La caractéristique de la propriété est introduite, mais son nom et son type sont génériques.*

- Lions écurie  
Une écurie  
explique  
entre  
cheval  
entre



# La structuration des éléments de modélisation

Autre exemple  
d'un élevage  
la liaison entre  
entre **Propriétaire**  
cheval de trait  
entre **caractéristique**  
entre **T** et **Int**



la gestion  
trait, d'où  
l'un  
n

- Les paquetages sont utiles pour structurer la modélisation d'un système d'information important. Un paquetage *système* contient les paquetages de plus haut niveau, eux-mêmes contenant d'autres paquetages et ainsi de suite, jusqu'aux éléments de base de la modélisation, comme les classes ou les cas d'utilisation. La relation de fusion et les paquetages template sont des éléments puissants pour structurer la modélisation d'un système d'information.

# Quizz





# Cycle de vie des objets

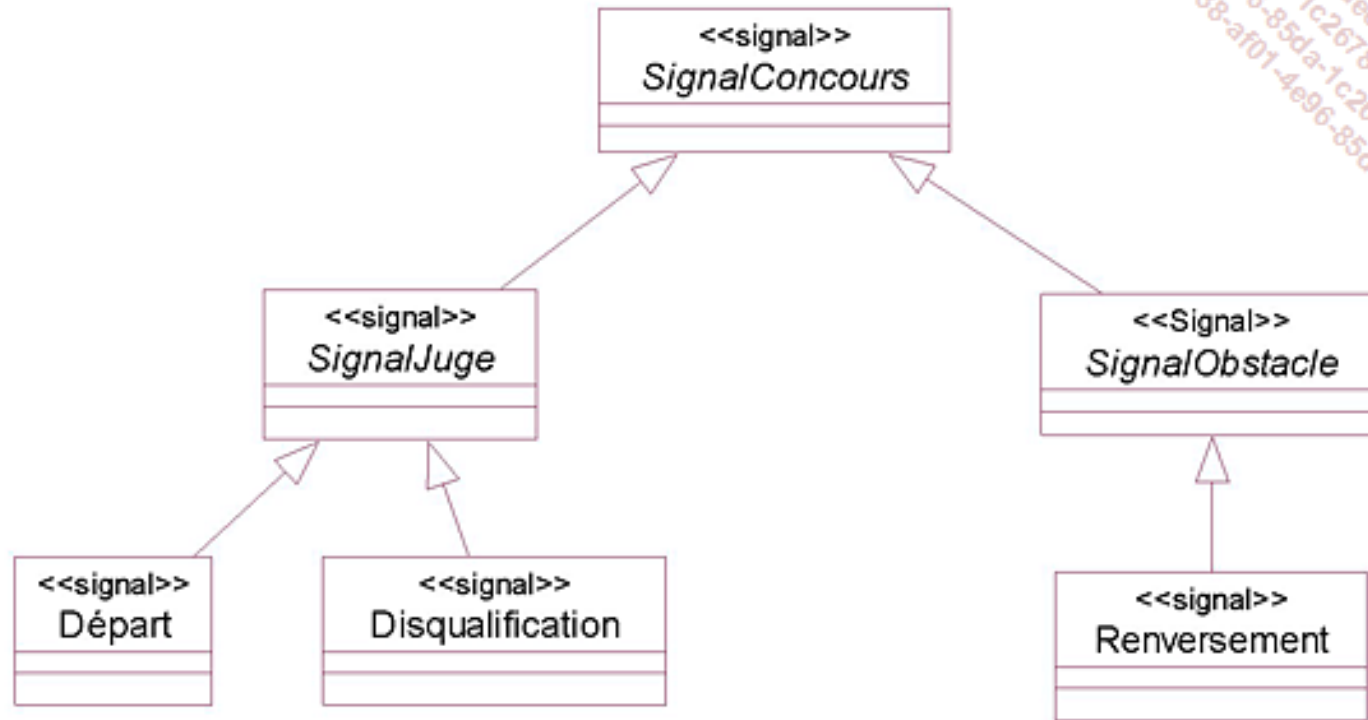
- L'état :
  - Moment du cycle de vie
    - Il peut juste attendre un signal (on dit qu'il est inactif)
    - Faire une activité ( on dit qu'il est actif)
      - Une activité est l'exécution d'une série de méthodes et d'interaction avec d'autres objets (voir diagramme d'activités)
  - Exemple : *Lors d'un concours de saut d'obstacles, le cheval est dans l'état de repos avant de commencer la compétition. Il s'agit d'un état où il est inactif et attend l'ordre de départ. Lorsqu'il saute un obstacle, le cheval est dans un état où il est actif et qui se termine lorsqu'il a fini de sauter l'obstacle.*
  - *Etat initial (après la création)*
  - *Un ou plusieurs états finaux ou de terminaison (destruction de l'objet)*
  - *Peut ne pas avoir d'état de fin (boucle perpétuelle)*

- Les événements : provoque un changement d'état
  - AnyReceivedEvent : correspond à un événement quelconque. Toutefois, il ne déclenche le changement d'état que si aucun autre événement dont le genre n'est pas AnyReceivedEvent n'enclenche au même instant un changement d'état. La syntaxe de cet événement est le mot-clé all.
  - CallEvent : un événement CallEvent se produit quand la méthode spécifiée est invoquée.
  - ChangeEvent : un événement ChangeEvent se produit quand un changement s'est produit dans le système. Ce changement est décrit par une expression logique qui devient vraie à l'instant où le changement se produit.

- TimeEvent : un événement TimeEvent est spécifié par une instruction indiquant un temps absolu ou relatif. Si le temps est relatif, l'instruction prend la syntaxe after expression. Si le temps est absolu, l'instruction prend la syntaxe at expression. Dans les deux cas, dès que le temps indiqué est atteint, l'événement se produit.
- SignalEvent : un événement SignalEvent se produit quand un signal est reçu.

- Les signaux sont des classes en UML
  - Signal émis ou reçus = instance de la classe signal
  - Ajout le stéréotype « signal » pour différencier
  - Attributs = paramètres du message
  - Organisation hiérarchique
  - UML n'impose pas la précision des événements
    - Au début souvent juste le nom

## Exemple des signaux recevable par un cavalier



On parle de réification

- La transition :
  - Lien orienté entre un état d'origine et un état de destination
  - Souvent associée à un événement
  - Peut avoir lieu que l'objet soit actif ou pas
    - Si passif : directement
    - Si actif : une fois l'activité associée finie

*Exemple : Lorsque le cavalier et le cheval reçoivent l'ordre de départ du juge, ils passent de l'état d'attente à l'état de course. Il s'agit d'un événement `SignalEvent`. S'il se produit n'importe quel autre événement, ils passent dans l'état de disqualification. Il s'agit alors d'un événement `AnyEvent`.*

*Un cheval reçoit le message de trotter. Il passe dans l'état où il trotte. Il s'agit d'un événement `CallEvent`. Un conseiller dans le monde équin a le statut de micro-entrepreneur. Quand son chiffre d'affaires annuel dépasse un montant fixé par la loi, il perd son statut de micro-entrepreneur.*

*Il s'agit d'un événement `ChangeEvent`. À 17 h 00, les chevaux de l'écurie sont lavés. Il s'agit d'un événement `TimeEvent`.*

- Une transition automatique n'est pas associée à un événement. Elle est franchie dès que l'objet a terminé l'activité liée à l'état d'origine. Dans ce cas, il est nécessaire d'associer une activité à l'état d'origine.

*Exemple : Lorsque le cavalier et le cheval ont terminé le parcours, ils passent automatiquement dans l'état final.*

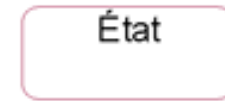


- Une transition réflexive possède le même état d'origine et de destination. Si la transition est associée à un événement, la réception de celui-ci ne fait pas changer l'état. Si la transition est réflexive et automatique, elle est alors utile pour réaliser une activité en boucle.

Exemple : *Tant que le cheval refuse de sauter un obstacle, il reste dans l'état de la course précédant cet obstacle. À chaque fois, le nombre de tentatives est augmenté de 1.*

- Il représente le cycle de vies des instances d'une classe
- Décrit états, les transitions qui les lient
- Décrit les événements qui provoquent les transitions
- Utile seulement si objet à un cycle de vie

- Etat : rectangle aux coins arrondis



- Premier Etat = état initial , il est unique



Un point noir

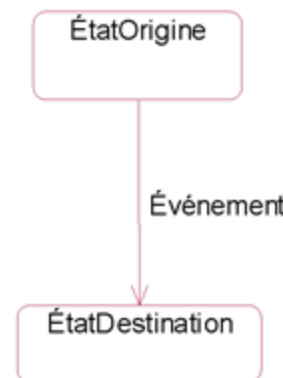
- Dernier Etat = état final, point noir entouré



- Etat de terminaison = fin du cycle de vie, une croix



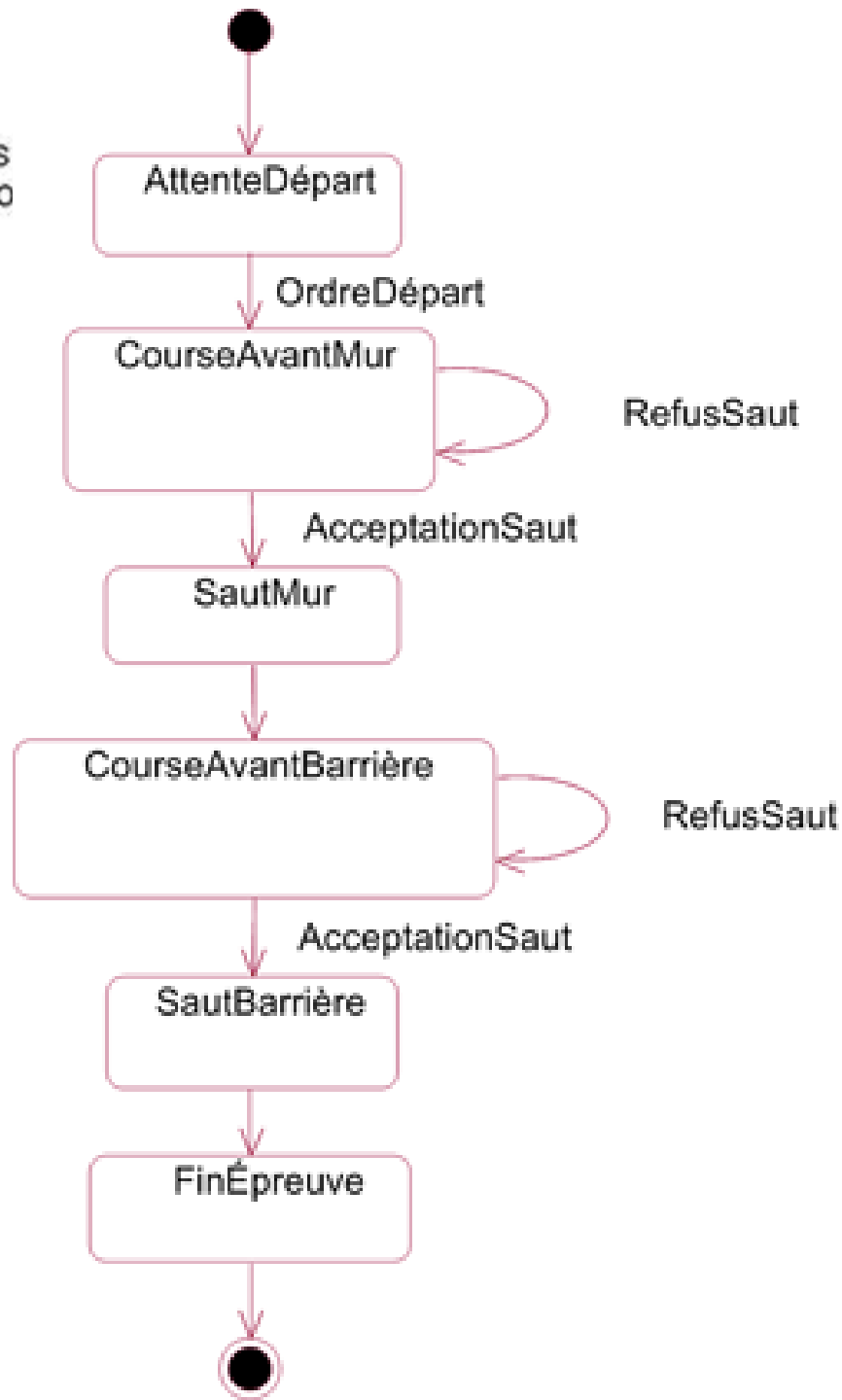
- Transition = trait droit fléché  
(ajout événement si utile)



- Transition réflexive

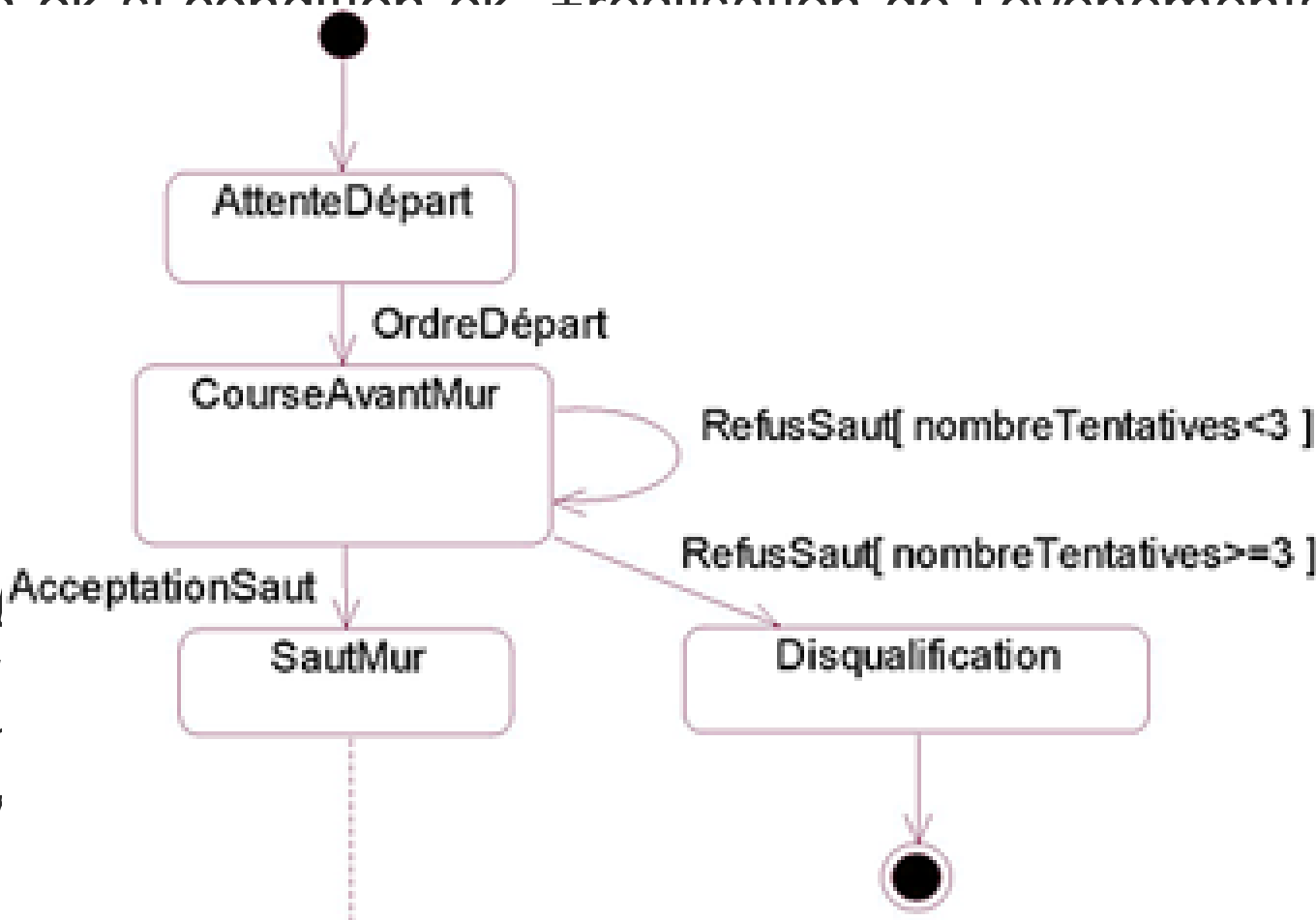


*Exemple : Dans un concours d'obstacles, l'épreuve consiste à demander à chaque concurrent de sauter deux ou trois obstacles différents. Il arrive que le cheval refuse de sauter un obstacle. Le concurrent peut alors recommencer le saut. La figure 8.8 représente le diagramme d'états-transitions décrivant une telle épreuve pour l'objet "concurrent de l'épreuve". Les deux obstacles sont respectivement le mur et la barrière. Ce diagramme contient des transitions réflexives et automatiques.*



- Transition ok si condition ok + réalisation de l'événement
- Se place

- Exemple  
a le droit  
après la  
figure illu  
en reprer  
est monti



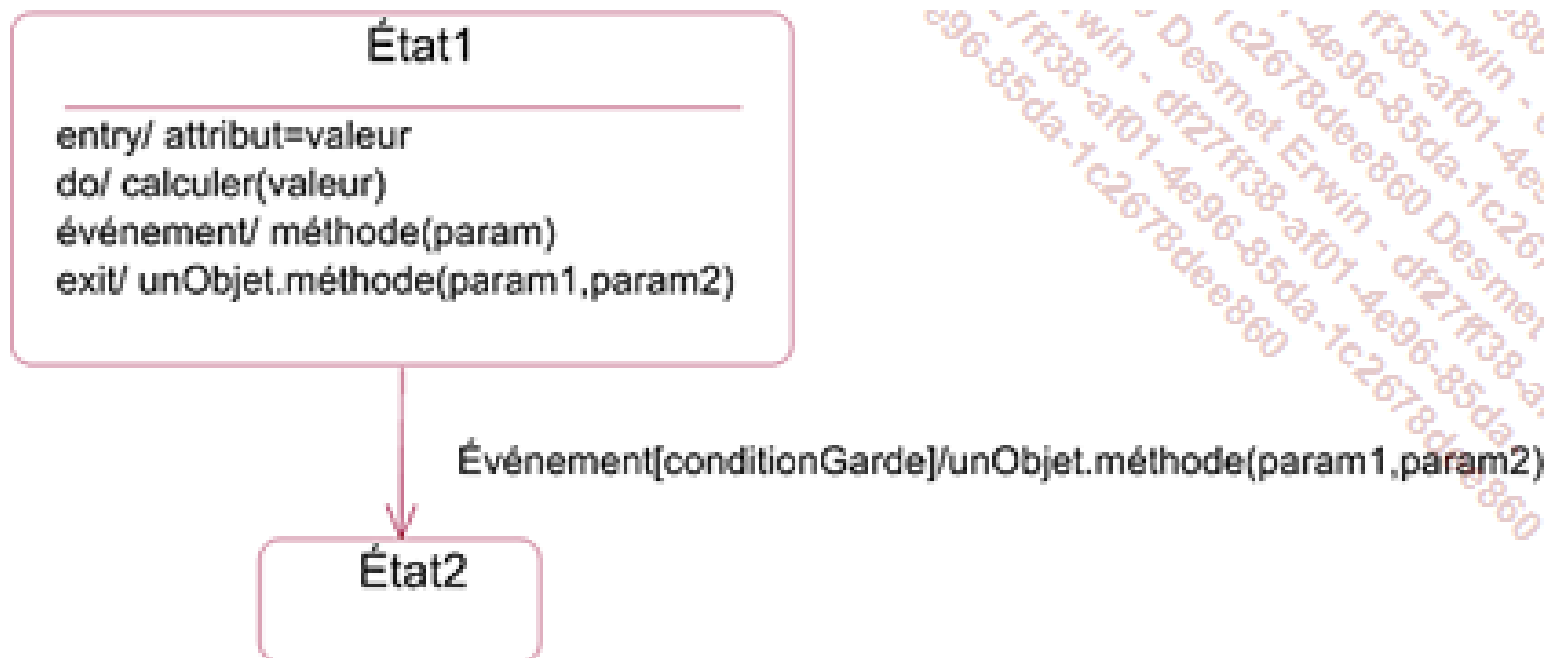
current  
à  
le refus  
obstacle

- Il est possible de spécifier différentes activités :
  - Pendant un état ;
  - Lors du franchissement d'une transition ;
  - À l'entrée et à la sortie d'un état ;
  - Au sein d'un état, lors de la réception d'un événement.
- Activité = série d'actions (attribuer valeur attribut, créer ou détruire objet, effectuer opération, invoquer méthode,...)

- Une activité précédée du mot-clé entry/ est exécutée lors de l'entrée dans l'état.
- Le mot-clé do/ introduit l'activité réalisée pendant l'état. Une activité précédée du nom d'un événement est exécutée si cet événement est reçu.
- Une activité précédée du mot-clé exit/ est exécutée lors de la sortie de l'état.
- Il est également possible de spécifier une activité lors du franchissement d'une transition. Dans ce cas, l'activité doit être précédée d'un /, à la suite de l'événement et de la condition de garde, s'ils existent.



## Elaborations du diagramme états-transitions : Activités liées



Exemple : *utilisation des activités au sein d'un état ou lors du franchissement d'une transition. Ceci permet notamment de gérer la valeur des attributs **nombrePointsPénalité** et **nombreTentatives** de la classe **Concurrent**. Le nombre de points de pénalité est augmenté si le mur est renversé, ce qui se traduit par la réception de l'événement **renversement** pendant l'état **SautMur**. Le nombre de tentatives est initialisé à 1, puis augmenté à chaque refus de sauter lors de la transition correspondante.*

