

# UML

De la modélisation métier aux spécifications logicielles avec UML



# Notre programme

- Introduction à UML
- Les concepts de l'approche par objets
- La modélisation des exigences
- La modélisation de la dynamique
- La modélisation des objets
- La structuration des éléments de modélisation
- La modélisation du cycle de vie des objets
- La modélisation des activités
- La modélisation de l'architecture du système

# **INTRODUCTION À UML**

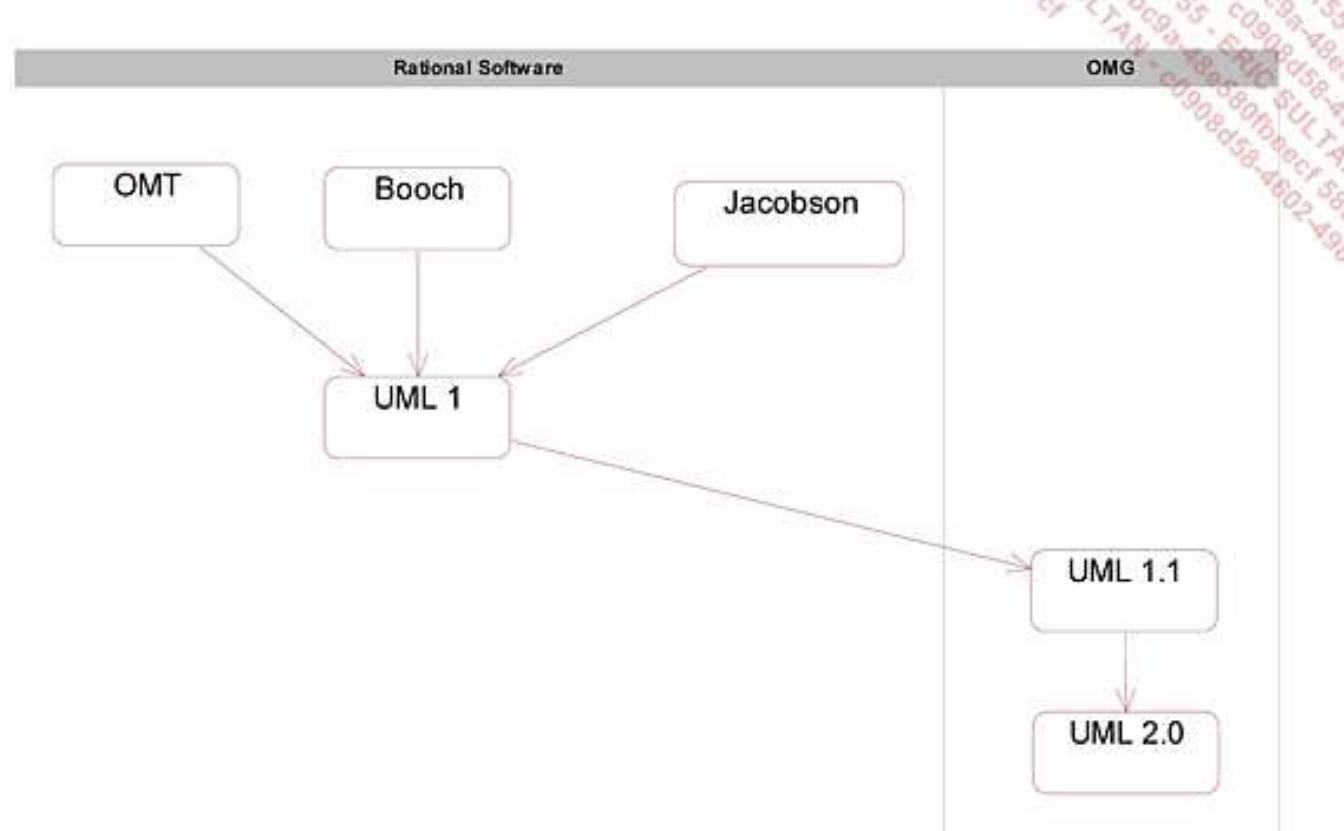
# UML ?

- UML (*Unified Modeling Language* ou langage unifié de modélisation) est un langage graphique
- UML est destiné à la modélisation de systèmes et de processus
- UML est un langage basé sur l'approche par objets
- UML est unifié car il provient de plusieurs notations qui l'ont précédé

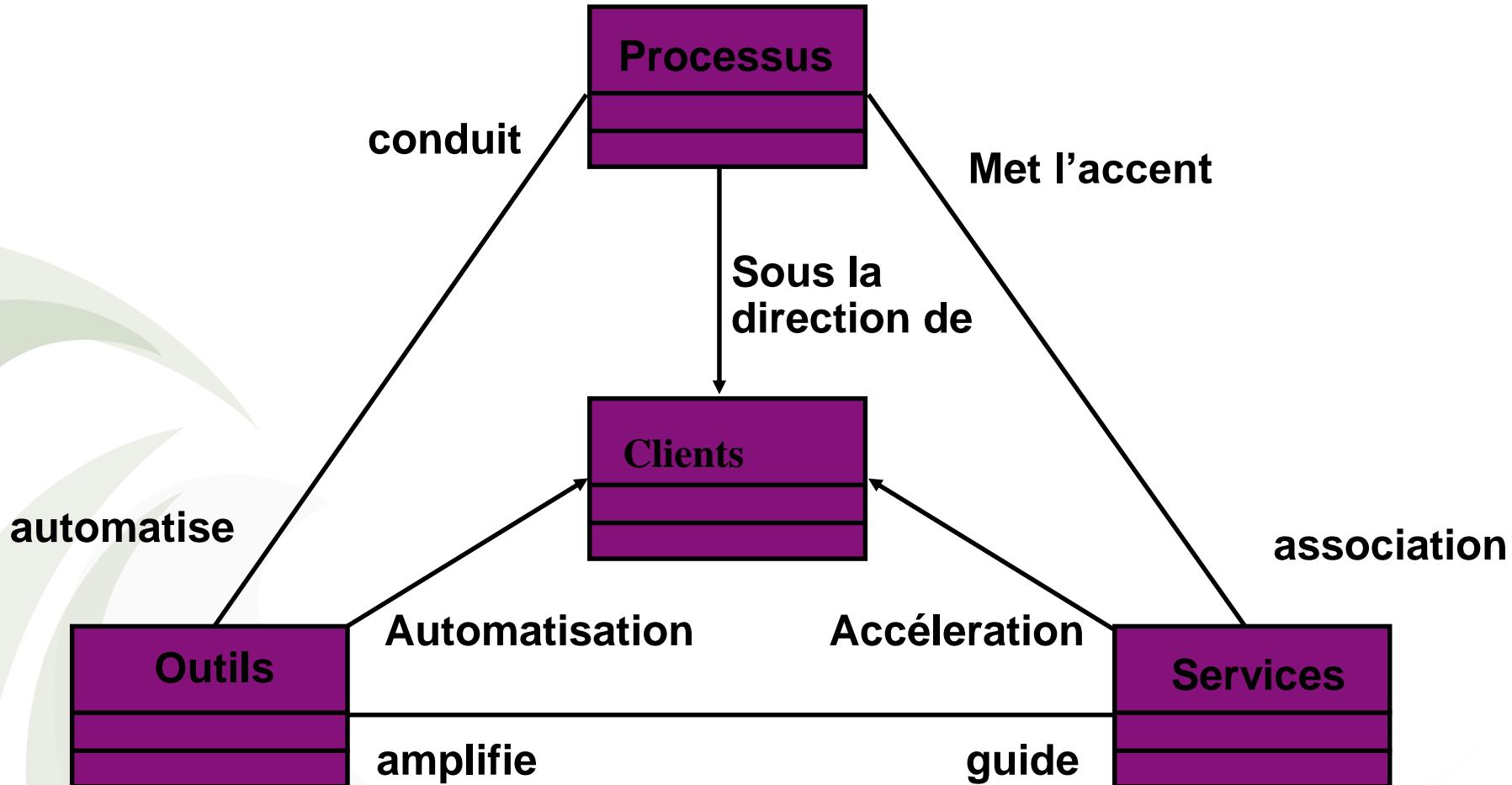
# Origine de l'objet et d'UML

- Langages de programmation objet : SIMULA67, SMALLTALK (1972) ...
- Modèle Entité-Association
- Réseaux sémantiques
- Méthodes structurés (SADT/SART, Merise, ...)
- Un grand nombre de méthodes orientées objet (OOSE, OBA, BOOCH, OMT, OOA, Martin-Odell, Classe-Relation, FUSION, ...)
- UML (Unified Modeling Language) , normalisé par l'OMG en novembre 1997, actuellement dans la version 2.0

# Origine de l'objet et d'UML



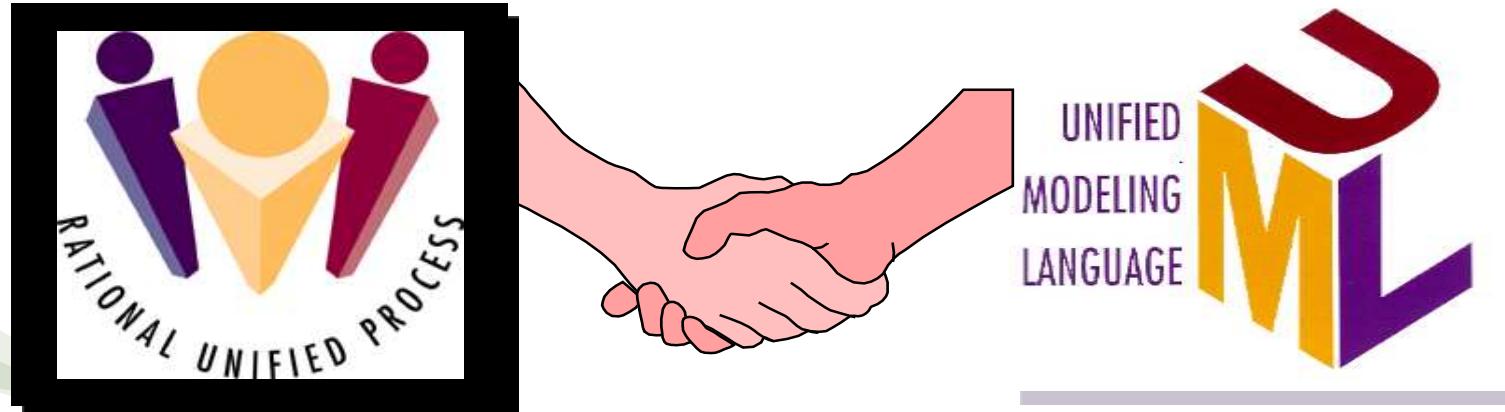
# RUP : Rational Unified Process



# Rational Unified Process (RUP)

- Unifie les meilleures pratiques de plusieurs disciplines dans un cycle de vie complet et consistant.
- Premier processus pour l'UML
- Intégré avec et supporté par les outils Rational
- Applicable à une variété d'applications et d'industries

# RUP et UML

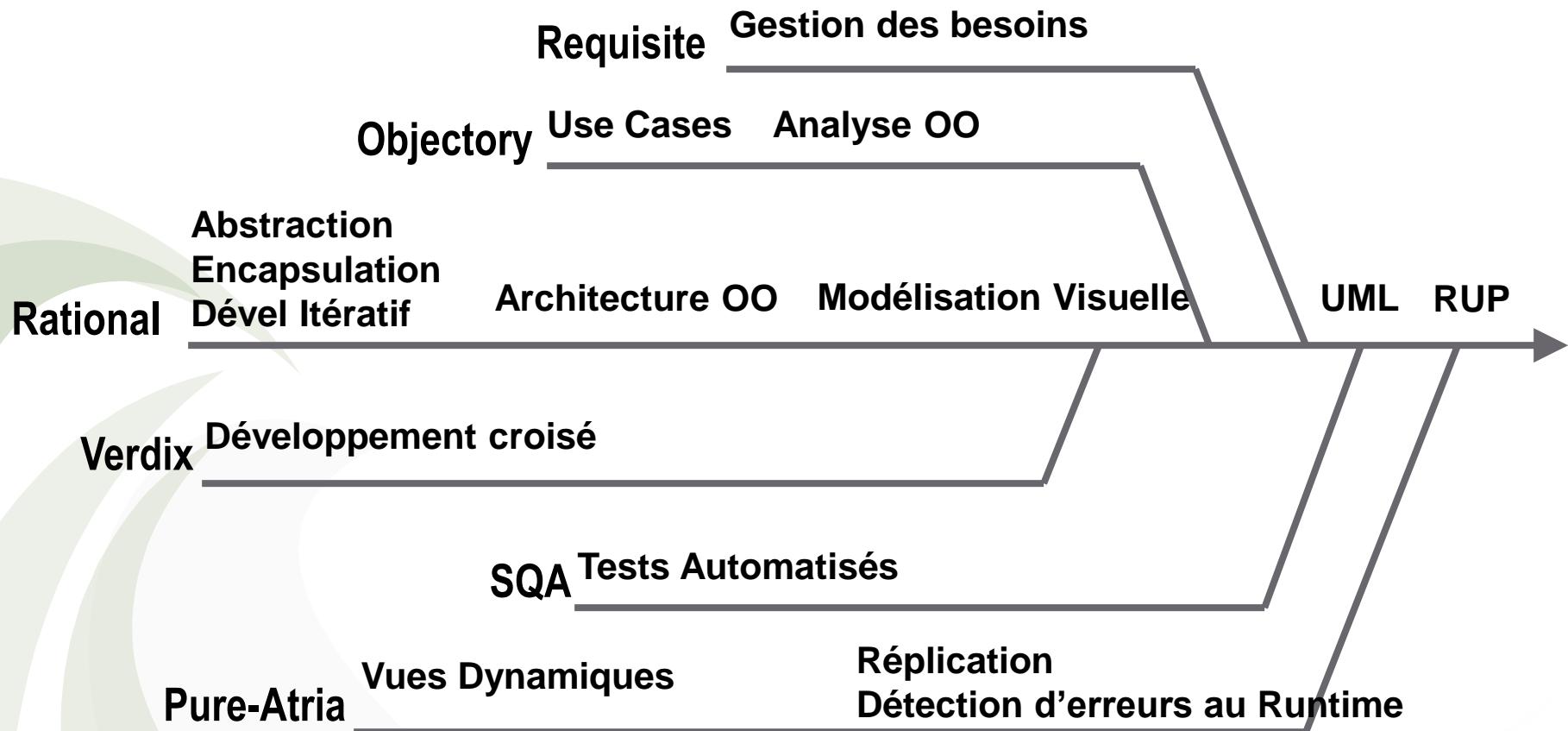


- Le Rational Unified Process et l'UML ont été développés main dans la main par Rational
- Contributions d'autres acteurs : Microsoft, HP, IBM, Oracle, Texas Instruments, MCI SystemHouse
- Standard au travers de l'OMG

# Rational Unified Process: Généologie

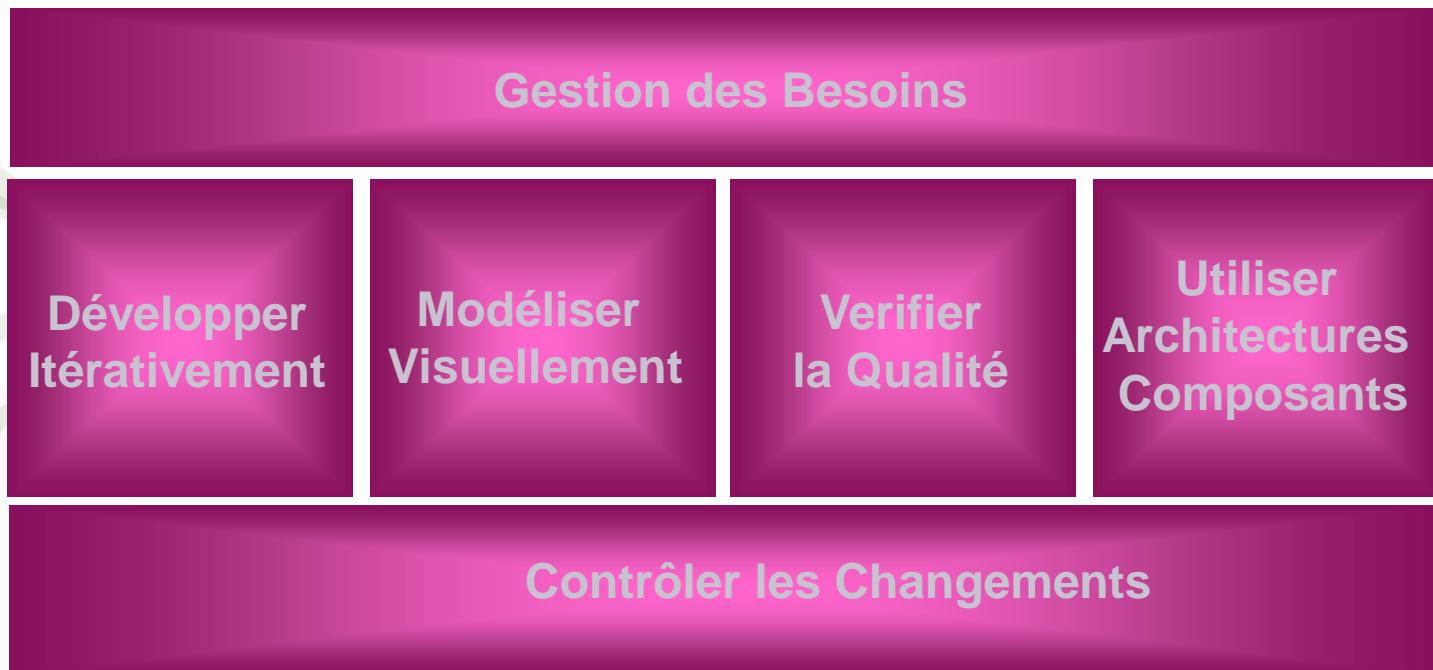
1981

1998

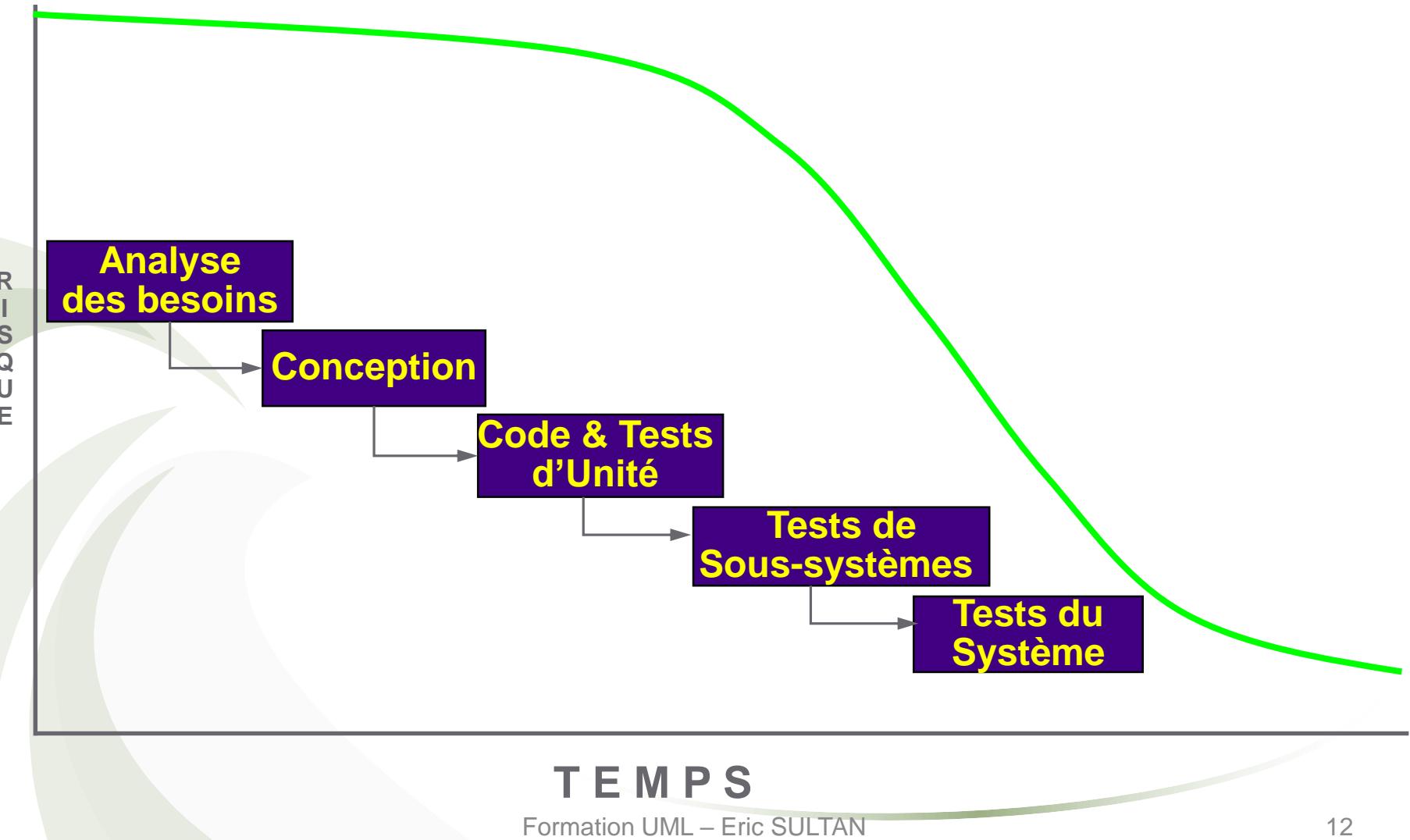


# Comment mettre en pratique le RUP ?

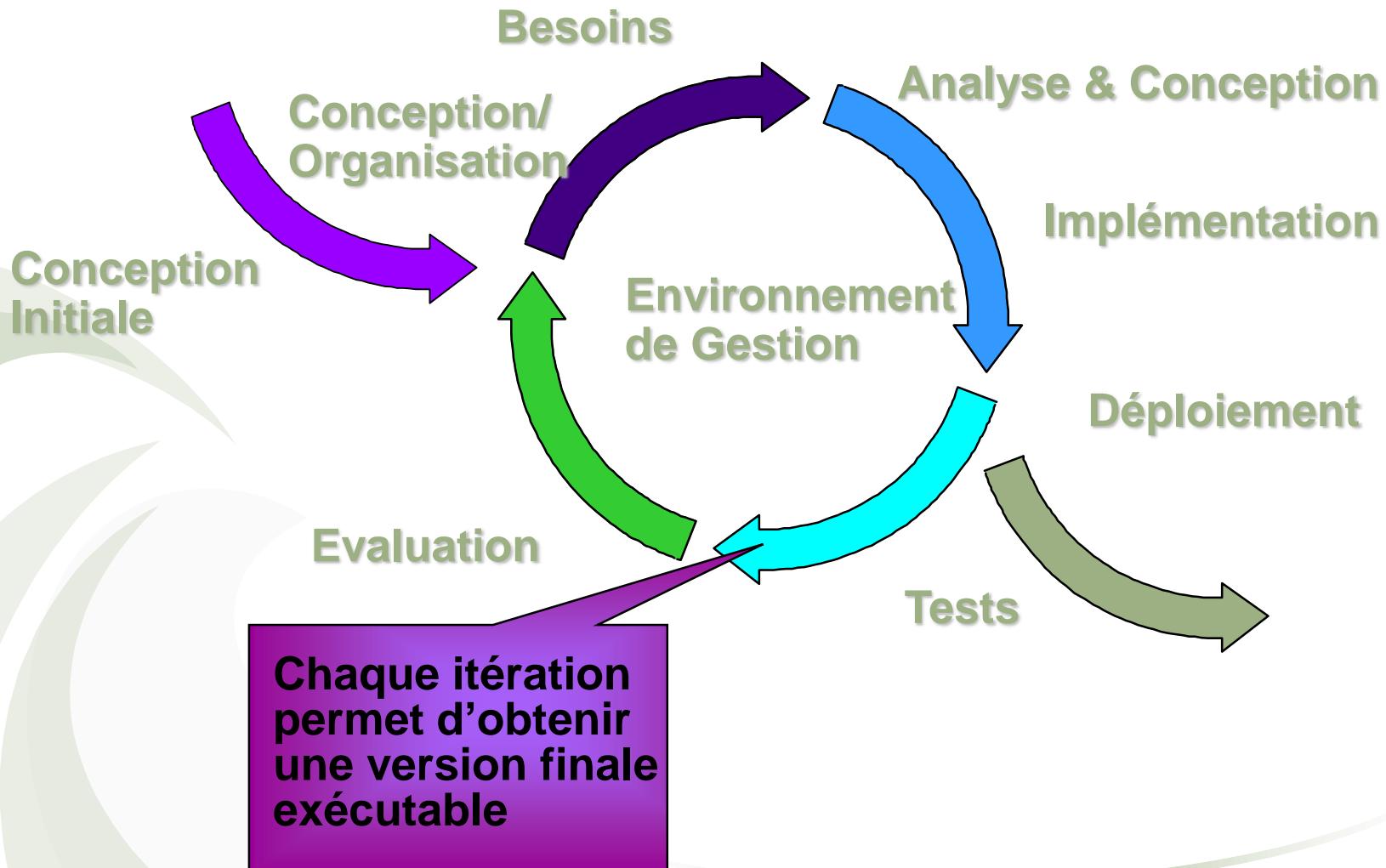
- Décrire l'implémentation effective des clés des “Meilleures Pratiques”



# Développement en cascade



# Développement Itératif



# Les principales activités de RUP

- La Modélisation des processus *métier*
- La gestion des exigences
- L'analyse et conception
- L'implantation et test
- Le déploiement

# MDA : Model Driven Architecture

- MDA est une nouvelle proposition de l'OMG dont l'objectif est la conception de systèmes basée sur la seule modélisation du domaine, en faisant abstraction des aspects technologiques.
- Dans MDA, le modèle des objets du domaine s'appelle PIM, c'est-à-dire *Platform Independent Model* ou modèle indépendant de la plateforme.
- Le lien avec UML réside au niveau du PIM.
- UML possède l'avantage de décrire finement des objets tout en restant indépendant des technologies.

# **LES CONCEPTS DE L'APPROCHE PAR OBJETS**

# Les concepts de l'approche par objets

- Découverte des différents concepts et principes de l'approche *objet* qui sont à la base d'UML
- Leur connaissance est indispensable pour comprendre les éléments utilisés dans la panoplie des diagrammes d'UML

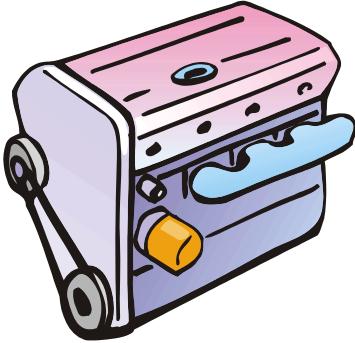
# Qu'est-ce qu'un objet ?

- Est objet tout ce qui peut être perçu, nommé, manipulé, ...
- Un objet se définit par :
  - son existence,
  - sa structure,
  - ses états,
  - son comportement.
- Exemples :
  - Ma jument Jorphée
  - Mon livre sur UML
  - L'article 293B du code des impôts

# Existence d'un objet

- L'existence d'un objet est ce qui lui est propre, ce qui le distingue de tout autre objet
- L'existence d'un objet est ce qui permet aux autres objets d'y faire référence pour obtenir sa collaboration
- Cette existence peut être identifiée par un ou différents noms propres
- Différents objets peuvent avoir le même nom
- Exemples :
  - *Un troupeau de chevaux est un système d'objets interagissant entre eux, chaque objet possédant son propre comportement*

# Structure d'un objet



- Un objet non primitif est une structure faite d'autres objets (ses composants)

# Événements

- Un événement est l'avènement d'une information
- Un objet peut réagir aux événements qu'il perçoit dans son environnement
- Les objets qui perçoivent un événement ne sont pas forcément connus de l'émetteur



# Invocation d'objets

- Les objets peuvent interagir explicitement en s'envoyant des messages, suivant un protocole déterminé.
- Les invocations d'objets diffèrent des appels procéduraux car :
  - elles fournissent toujours une référence à l'objet auquel le message est envoyé
  - la réaction de l'objet destinataire dépend de celui-ci et de son état



# Comportement

- Le comportement d'un objet est défini par les actions qu'il entreprend dès sa création ou suite à la perception d'un événement ou la réception d'un message
- Le comportement d'un objet peut entraîner la création ou la destruction d'autres objets ou des changements d'états
- Un objet est dit « actif » lorsqu'il entreprend et contrôle des actions dès sa création et/ou à partir d'événements qu'il perçoit dans son environnement (et non pas seulement sur réception de messages d'autres objets)

# Concurrence

- Différents objets peuvent exécuter leur comportement en parallèle.
- Si ces objets se partagent une même ressource, ils sont dits « concurrents » : un objet peut être bloqué jusqu'à ce qu'un autre objet ait terminé son activité.
- Différents types de synchronisation permettent de décrire les modalités d'interaction des objets.
- La notion de concurrence peut s'appliquer aux activités parallèles d'un même objet.

# L'abstraction

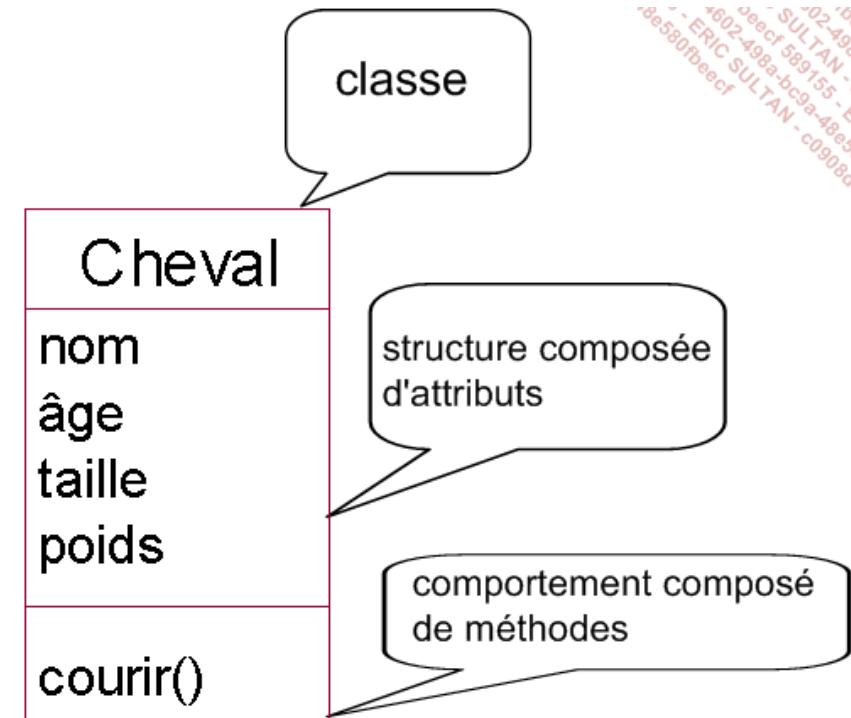
- L'abstraction est un principe très important en modélisation
- Elle consiste à retenir uniquement les propriétés pertinentes d'un objet pour un problème précis
- Les objets utilisés en UML sont des abstractions d'objets du monde réel
- Exemples :
  - *On s'intéresse aux chevaux pour l'activité de course. 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 similaires, c'est-à-dire possédant la même structure et le même comportement et constitués des mêmes attributs et méthodes, forme une classe d'objets
- La structure et le comportement peuvent alors être définis en commun au niveau de la classe
- Chaque objet d'une classe, encore appelé instance de classe, se distingue par son identité propre et possède des valeurs spécifiques pour ses attributs

# Les classes d'objets

- *L'ensemble des chevaux constitue la classe Cheval qui possède la structure et le comportement*



# Les classes d'objets

- *Le cheval Jorphée est une instance de la classe Cheval dont les attributs et leurs valeurs sont illustrés*

Jorphée : Cheval

nom = Jorphée  
âge = 8  
taille = 1,72  
poids = 550

valeurs

# Les classes d'objets

- Le nom d'une classe apparaît au singulier
- Il est toujours 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

# L'encapsulation

- L'encapsulation consiste à masquer des attributs et des méthodes de l'objet vis-à-vis des autres objets
- En effet, certains attributs et méthodes ont pour seul objectif des traitements internes à l'objet et ne doivent pas être exposés aux objets extérieurs
- Encapsulés, ils sont appelés les attributs et méthodes privés de l'objet

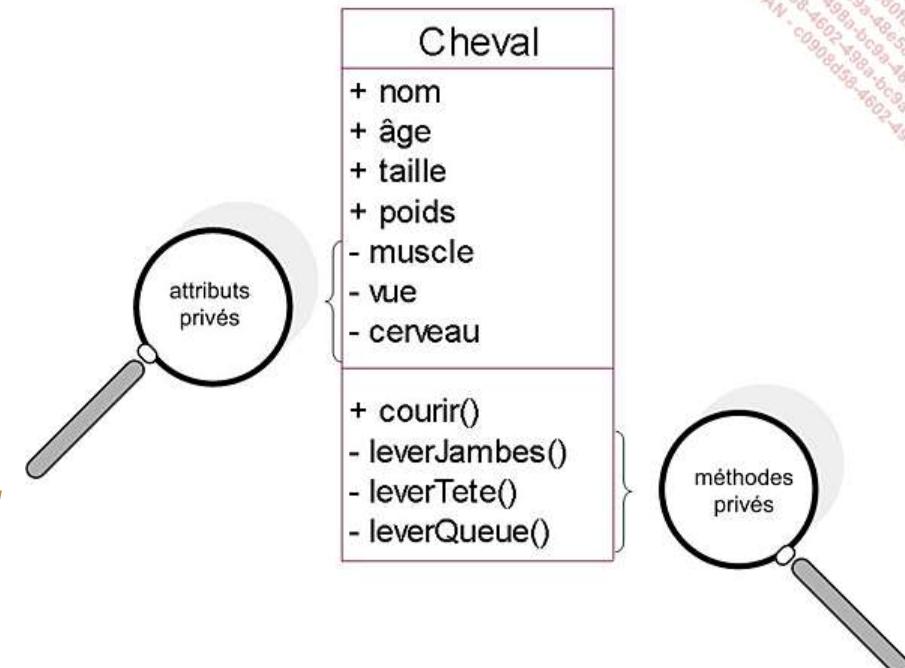
# L'encapsulation

- L'encapsulation est une abstraction puisque l'on simplifie la représentation de l'objet vis-à-vis des objets extérieurs
- Cette représentation simplifiée est constituée des attributs et méthodes publiques de l'objet
- Dans la notation UML, les attributs et méthodes publics sont précédés du signe plus tandis que les attributs et méthodes privés (encapsulés) sont précédés du signe moins

# L'encapsulation

## □ Exemple :

- *Lorsqu'il court, un cheval va effectuer différents mouvements comme lever les jambes, lever la tête, lever la queue. Ces mouvements sont internes au fonctionnement de l'animal et n'ont pas à être connus à l'extérieur. Ce sont des méthodes privées. Ces opérations accèdent à une partie interne du cheval : ses muscles, son cerveau et sa vue. Cette partie interne est représentée sous la forme d'attributs privés*



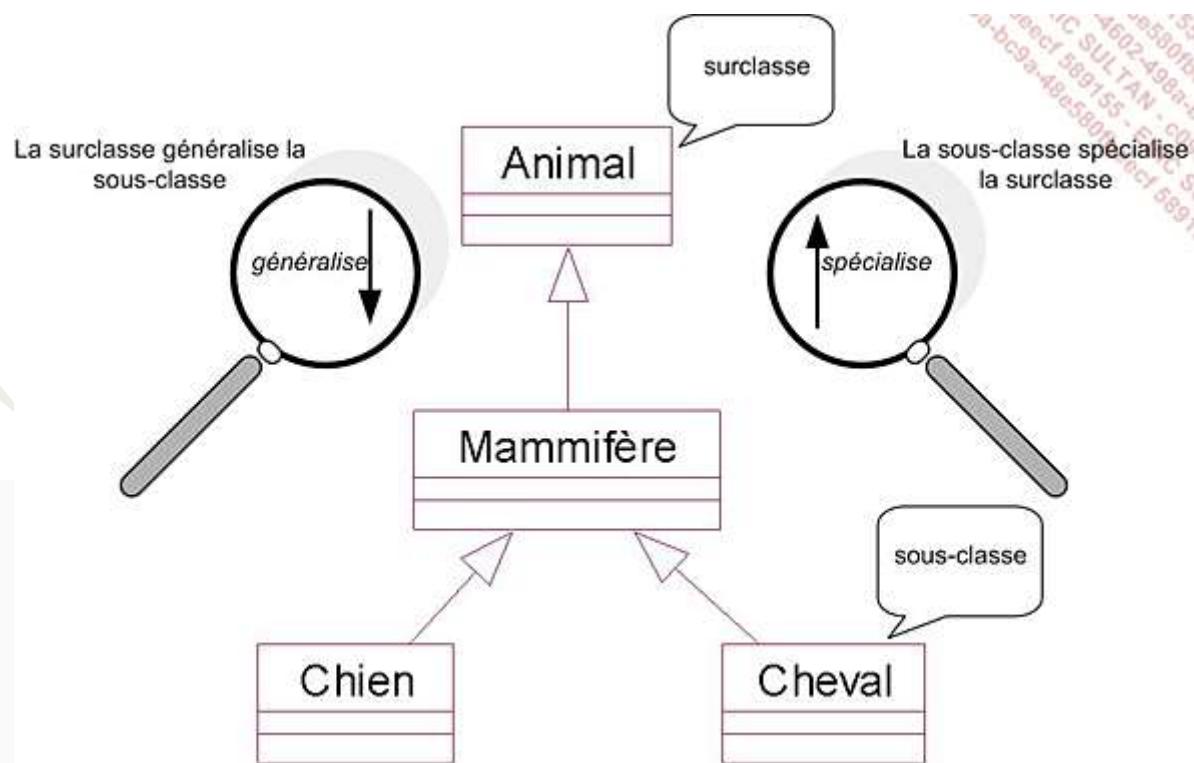
# La spécialisation et la généralisation

- Jusqu'à présent, chaque classe d'objets est introduite séparément des autres classes
- Il s'agit alors d'une sous-classe d'une autre classe
- Elle constitue ainsi une spécialisation de cette autre classe
- Exemple :
  - *La classe des chevaux est une sous-classe de la classe des mammifères*
- La généralisation est la relation inverse de la spécialisation

# La spécialisation et la généralisation

- Si une classe est une spécialisation d'une autre classe, cette dernière est une généralisation de la première. Elle en est sa surclasse
- Exemple :
  - *La classe des mammifères est une surclasse de la classe des chevaux*
- La relation de spécialisation peut s'appliquer à plusieurs niveaux, donnant lieu à une hiérarchie de classes
- Exemple :
  - *La classe des chevaux est une sous-classe de la classe des mammifères, elle-même sous-classe de la classe des animaux. La classe des chiens est une autre sous-classe de la classe des mammifères. La hiérarchie correspondante des classes est représentée par ...*

# La spécialisation et la généralisation



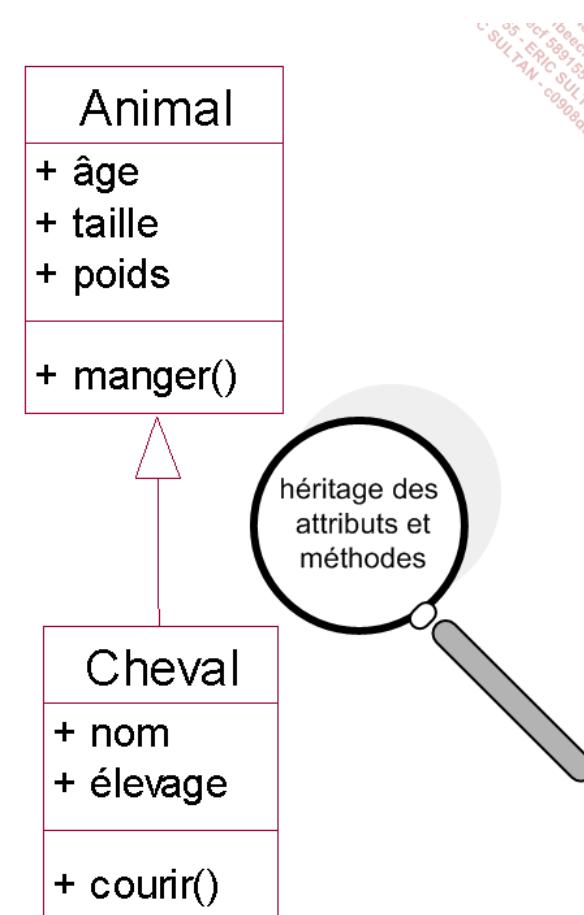
# L'héritage

- L'héritage est la propriété qui fait bénéficier à une sous-classe de la structure et du comportement de sa surclasse
- L'héritage provient du fait qu'une sous-classe est un sous-ensemble de sa surclasse
- En conséquence, elles bénéficient :
  - de la structure et du comportement définis dans cette surclasse
  - en plus de la structure et du comportement introduits au niveau de la sous-classe

# L'héritage

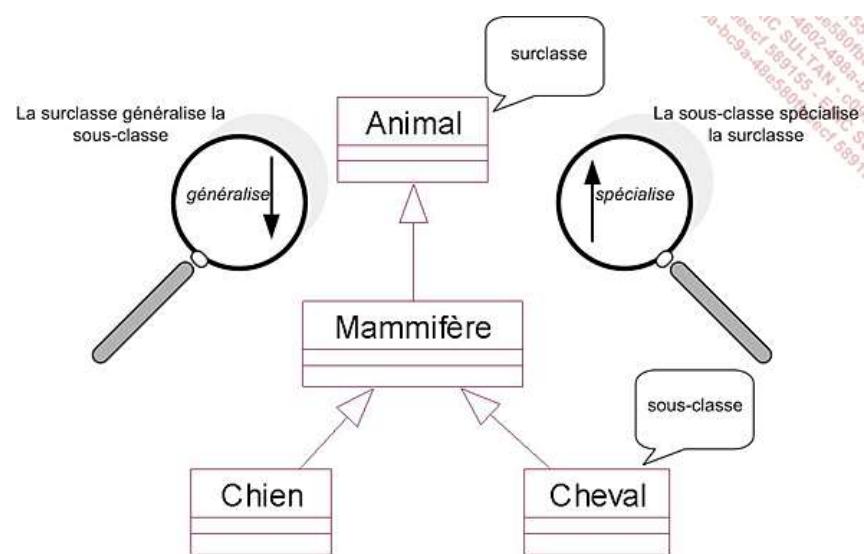
- Exemple :

- Soit un système où la classe *Cheval* est une sous-classe directe de la classe *Animal*, un cheval est alors décrit par la combinaison de la structure et du comportement issus des classes *Cheval* et *Animal*, c'est-à-dire avec les attributs *âge*, *taille*, *poids*, *nom* et *élevage* ainsi que les méthodes *manger* et *courir*



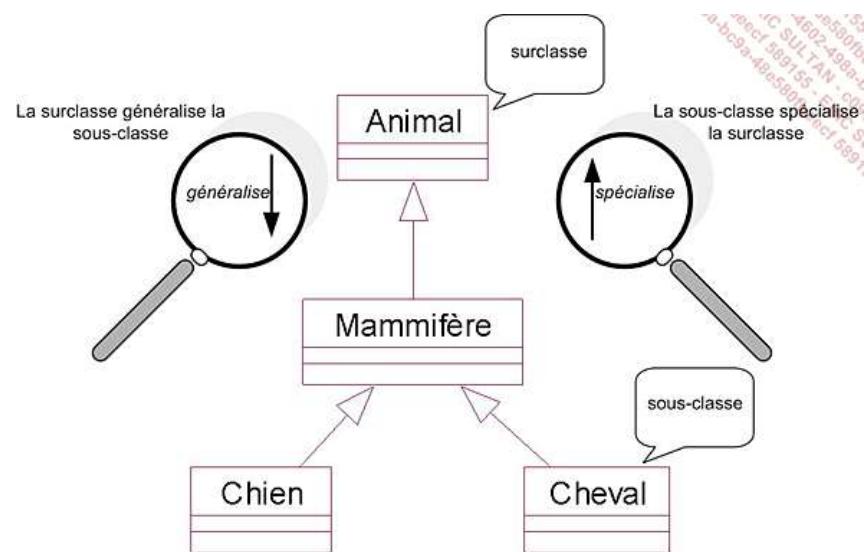
# Les classes abstraites et concrètes

- L'examen de la hiérarchie présentée montre qu'il existe deux types de classes dans la hiérarchie



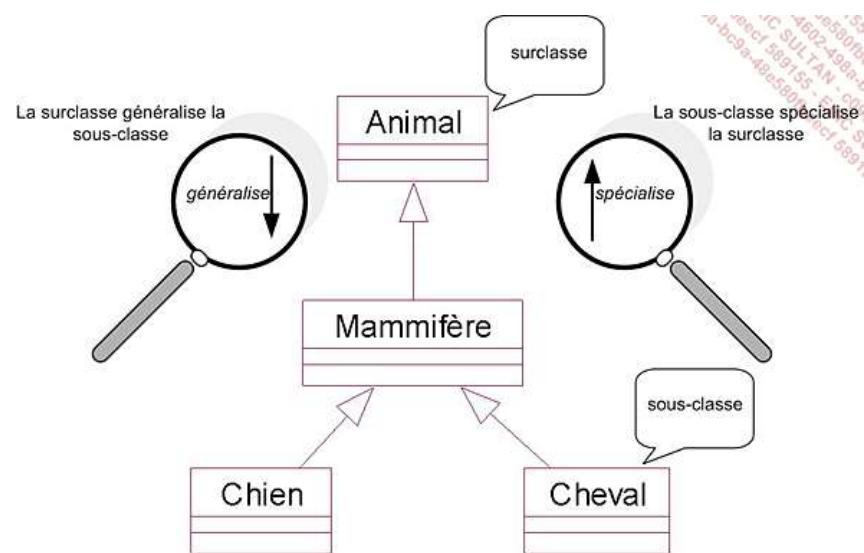
# Les classes abstraites et concrètes

- Des classes qui possèdent des instances, à savoir les classes *Cheval* et *Chien*.
- Ces classes sont appelées classes concrètes



# Les classes abstraites et concrètes

- Des classes qui n'en possèdent pas directement, comme la classe *Animal*
- En effet, si dans le monde réel, il existe des chevaux, des chiens, le concept d'animal reste, quant à lui, abstrait
- La classe *Animal* est appelée une classe abstraite



# Les classes abstraites et concrètes

- Une classe abstraite a pour vocation de posséder des sous-classes concrètes
- Elle sert à factoriser des attributs et des méthodes communs à ses sous-classes
- En UML, le nom des classes abstraites apparaît en caractères italiques

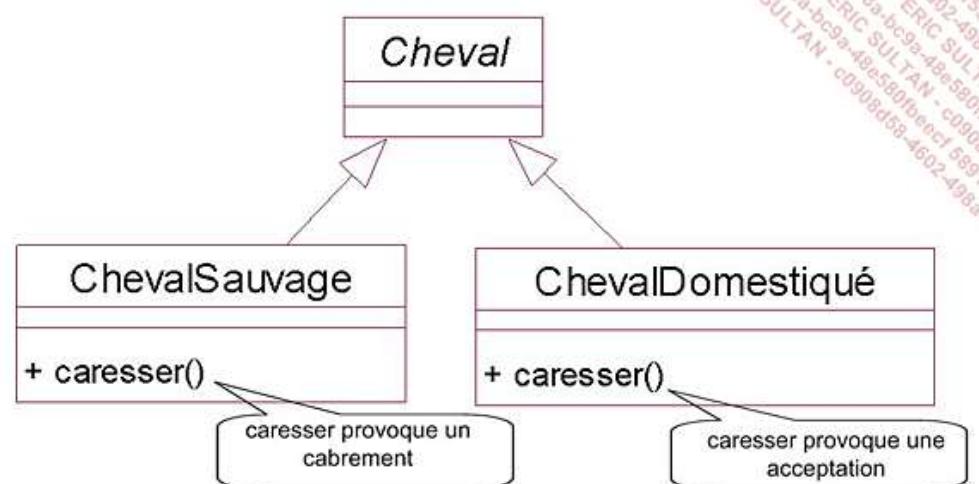
# Le polymorphisme

- Le polymorphisme signifie qu'une classe (très généralement abstraite) représente un ensemble constitué d'objets différents car ils sont instances de sous-classes distinctes
- Lors de l'appel d'une méthode de même nom, cette différence se traduit par des comportements différents (sauf dans le cas où la méthode est commune et héritée de la surclasse dans les sous-classes)

# Le polymorphisme

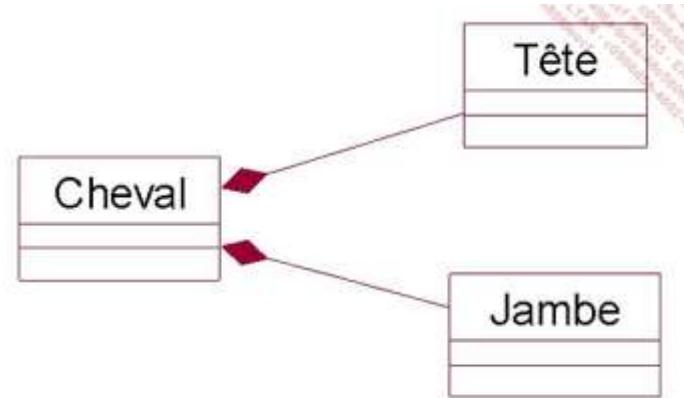
## □ Exemple :

- La méthode caresser a un comportement différent selon que le cheval est instance de ChevalSauvage ou de ChevalDomestiqué
- Dans le premier cas, le comportement sera un refus (se traduisant par un cabrement) alors que dans le second, le comportement sera une acceptation
- Si l'on considère la classe Cheval dans son intégralité, on a donc un ensemble de chevaux qui ne réagissent pas de la même façon lors de l'activation de la méthode caresser



# La composition

- Un objet peut être complexe et composé d'autres objets
- L'association qui unit alors ces objets est la composition
- Elle se définit au niveau de leurs classes mais les liens sont bâtis entre les instances des classes
- Les objets formant l'objet composé sont appelés *composants*
- *Exemple :*
  - *Un cheval est un exemple d'objet complexe. Il est constitué de ses différents organes (jambes, tête, etc.)*



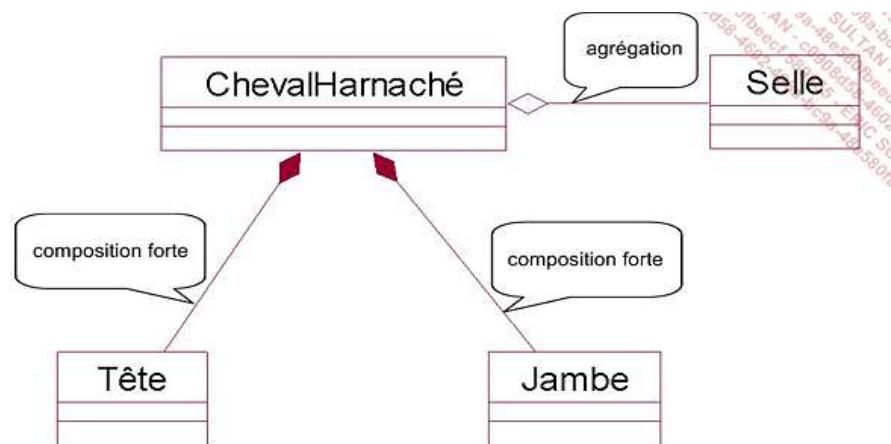
# La composition

- ❑ La composition peut prendre deux formes :
  - la composition faible ou agrégation
  - la composition forte
- ❑ Dans la composition faible, les composants peuvent être partagés entre plusieurs objets complexes
- ❑ Dans la composition forte, les composants ne peuvent être partagés et la destruction de l'objet composé entraîne la destruction de ses composants

# La composition

## □ Exemple :

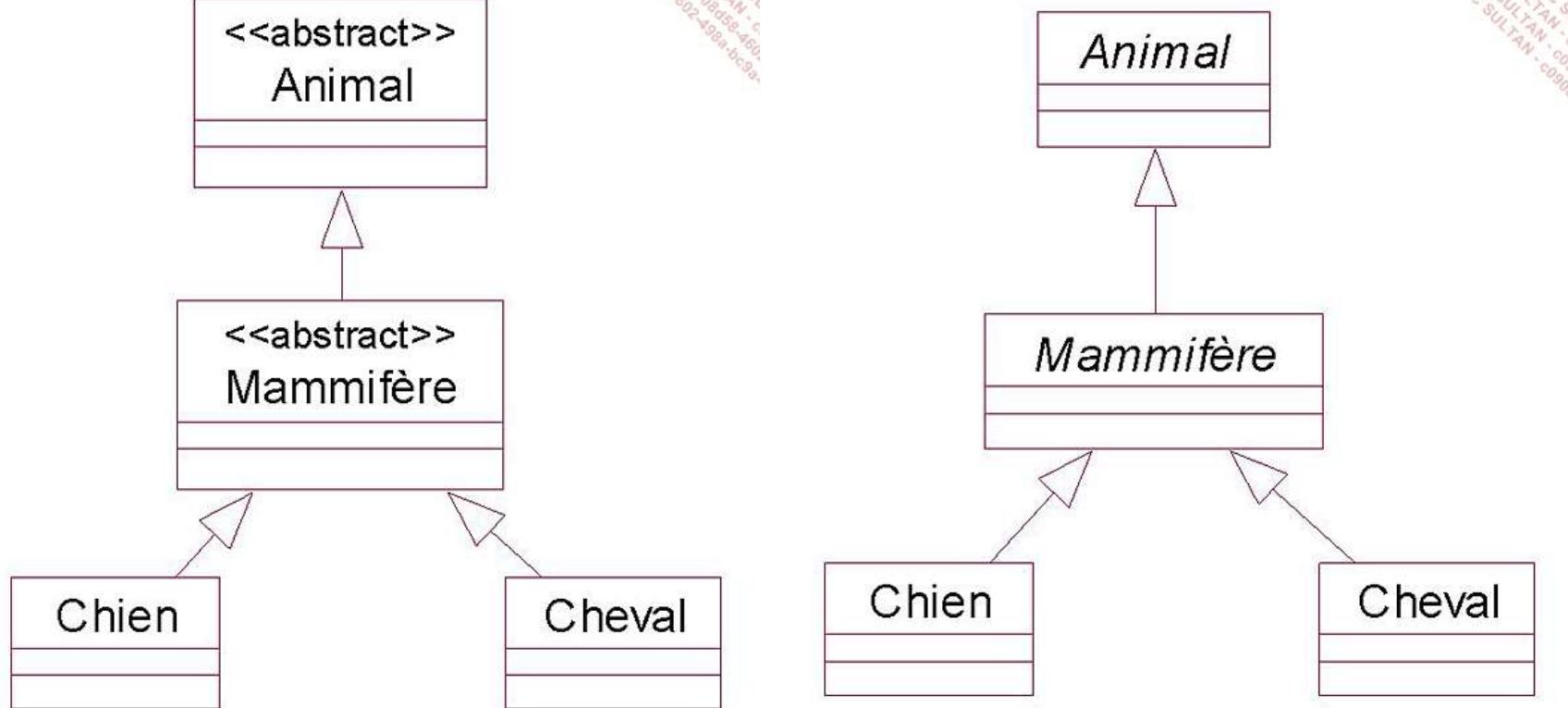
- Si l'on reprend l'exemple précédent dans le cas d'un cheval de course harnaché et si l'on ajoute la selle dans les composants, on obtient :
  - Une composition forte pour les jambes et la tête ; en effet, jambes et tête ne peuvent pas être partagées et la disparition du cheval entraîne la disparition de ses organes
  - Une agrégation ou composition faible pour la selle



# La spécialisation des éléments : la notion de stéréotype en UML

- Un stéréotype est constitué d'un mot clé explicitant cette spécialisation
- Celui-ci est noté entre guillemets
- Cette spécialisation est réalisée indépendamment du système que l'on cherche à modéliser
- Exemple :
  - *Le concept de classe abstraite est un concept spécialisé du concept de classe*
  - *Nous avons vu qu'une classe abstraite est représentée comme une classe avec un nom en italiques*
  - *Cette représentation graphique inclut un stéréotype implicite, mais il est également possible de ne pas mettre le nom de la classe en italiques et de préciser explicitement le stéréotype «abstract»*

# La spécialisation des éléments : la notion de stéréotype en UML



# Conclusion

- L'approche par objets forme la base d'UML
- Elle est constituée de concepts (objets, classes, spécialisation, composition) et de principes (abstraction, encapsulation)
- Cet ensemble fait de l'approche par objets un véritable support pour la modélisation de systèmes complexes, et au-delà d'UML, pour leur programmation

# **LA MODÉLISATION DES EXIGENCES**

# Introduction

- Découverte des cas d'utilisation employés pour décrire les exigences fonctionnelles attendues, lors de la rédaction du cahier des charges d'un système, ou les fonctionnalités d'un système existant
- L'ensemble des cas d'utilisation d'un système contient les exigences fonctionnelles attendues ou existantes, les acteurs (utilisateurs du système) ainsi que les relations qui unissent acteurs et fonctionnalités
- Cet ensemble détermine également les frontières du système, à savoir les fonctionnalités remplies par le système et celles qui lui sont externes

# Introduction

- Les cas d'utilisation servent de support pour les étapes de modélisation, de développement et de validation
- Ils constituent un référentiel du dialogue entre les informaticiens et les clients et, par conséquent, une base pour l'élaboration au niveau fonctionnel du cahier des charges

# Cas d'utilisation

- Les cas d'utilisation décrivent sous la forme d'actions et de réactions, le comportement du système étudié du point de vue des utilisateurs
- Ils définissent les limites du système et ses relations avec son environnement
- Entre un utilisateur et le système, un cas d'utilisation décrit les interactions liées à un objectif fonctionnel de l'utilisateur
- Exemple :
  - *Considérons comme système un élevage de chevaux. L'achat d'un cheval par un client constitue un cas d'utilisation*

# Acteur

- Un utilisateur externe du système peut jouer différents rôles vis-à-vis du système
- Un couple (utilisateur, rôle) constitue un acteur spécifique désigné en UML uniquement par le nom du rôle
- Cette définition est étendue aux autres systèmes qui interagissent avec le système; ils forment autant d'acteurs qu'ils jouent de rôle

# Acteur

- Deux catégories d'acteurs doivent être distinguées :
  - les acteurs primaires, pour lesquels l'objectif du cas d'utilisation est essentiel
  - les acteurs secondaires qui interagissent avec le cas d'utilisation mais dont l'objectif n'est pas essentiel
- 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.*

# Scénario

- Un scénario est une instance d'un cas d'utilisation dans laquelle toutes les conditions relatives aux différents événements ont été fixées
- À un cas d'utilisation donné correspondent plusieurs scénarios
- 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 Jorphée par Fien constitue un exemple de scénario du cas d'utilisation d'achat d'un cheval*
  - *Toutes les alternatives du déroulement sont connues, car Fien a acquis Jorphée*

# Relation de communication

- La relation qui lie un acteur à un cas d'utilisation s'appelle la relation de communication
- Cette relation supporte différents modèles de communication, par exemple :
  - les services que le système doit fournir à chacun des acteurs du cas d'utilisation
  - les informations du système qu'un acteur peut introduire, consulter ou modifier
  - les changements intervenant dans l'environnement dont un acteur informe le système
  - les changements intervenant au sein du système dont ce dernier informe un acteur

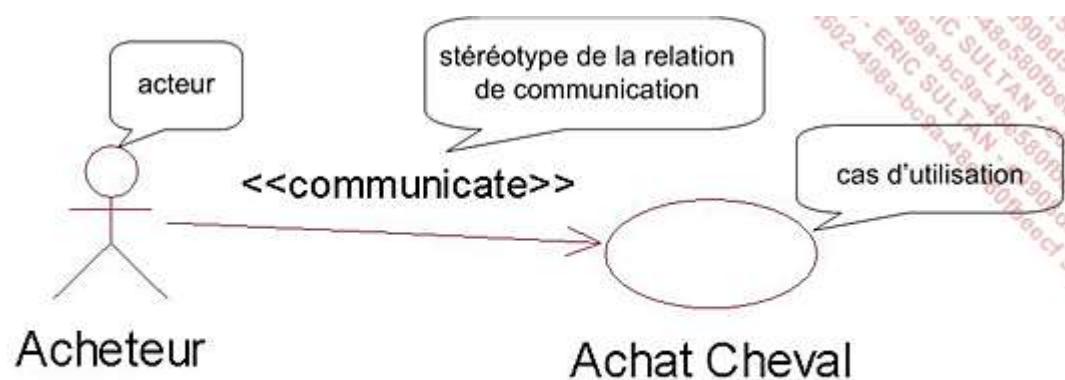
# Relation de communication

## □ Exemple :

- *Lorsque Fien a acquis Jorphée, elle a reçu des informations de l'élevage comme la proposition de prix, les papiers de la jument, son carnet de vaccination et a fourni des informations comme une contre-proposition de prix, une promesse d'achat.*

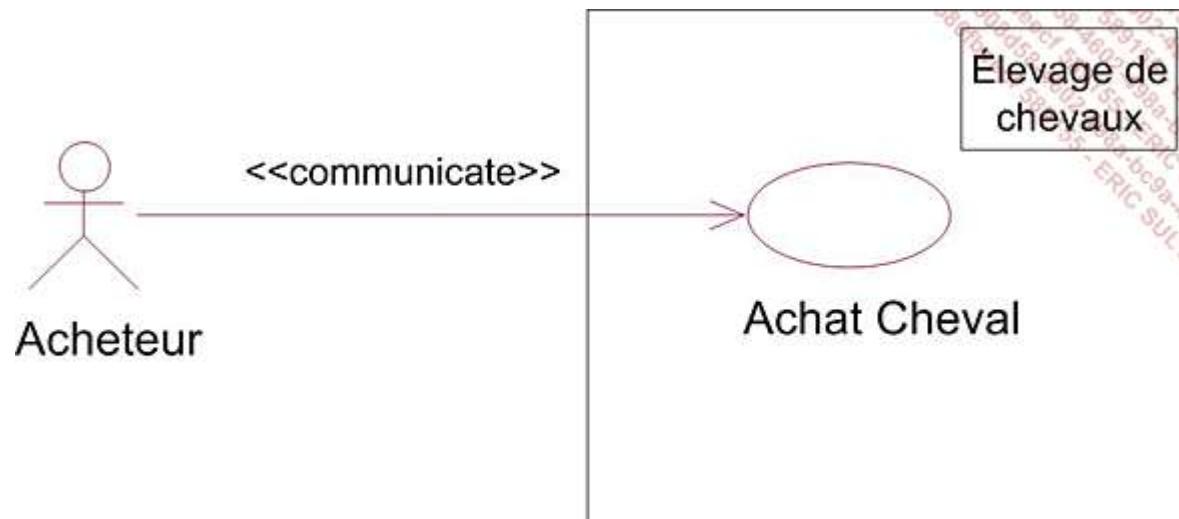
# Le diagramme des cas d'utilisation

- Le diagramme des cas d'utilisation montre les cas d'utilisation représentés sous la forme d'ovales et les acteurs sous la forme de personnages
- Il indique également les relations de communication qui les relient
- Exemple :
  - *Le cas d'utilisation de l'achat d'un cheval*



# Le diagramme des cas d'utilisation

- Il est possible de représenter le système qui répond au cas d'utilisation sous la forme d'un rectangle englobant le cas
- Exemple :



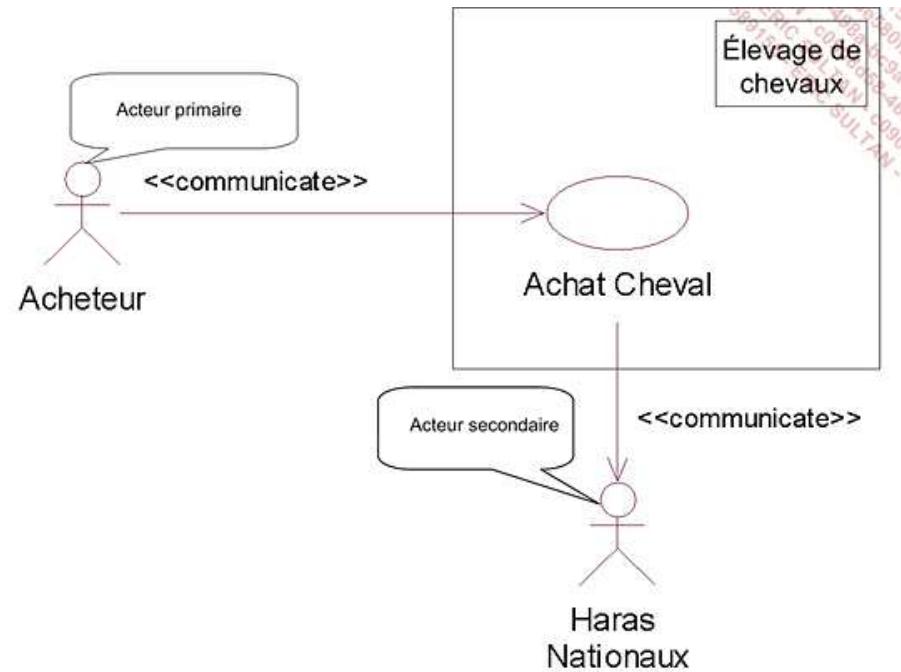
# Le diagramme des cas d'utilisation

- Un acteur secondaire est représenté comme un acteur primaire
- Souvent, le sens de la relation de communication entre un acteur secondaire et le système est inversé par rapport au sens de la relation entre un acteur primaire et le système
- En effet, la communication est initiée par le système et non par l'acteur

# Le diagramme des cas d'utilisation

## □ Exemple :

- *Le changement de propriétaire du cheval est réalisé par les haras nationaux*



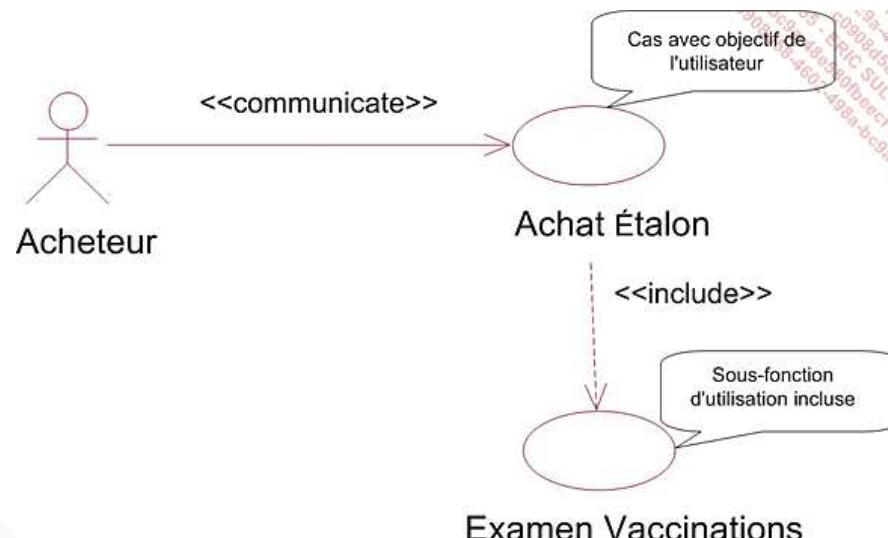
# Les relations entre les cas d'utilisation

## □ La relation d'inclusion

- La relation d'inclusion sert à enrichir un cas d'utilisation par un autre cas d'utilisation
- Cet enrichissement est réalisé par une inclusion impérative, il est donc systématique
- Un tel cas d'utilisation est une sous-fonction
- L'inclusion sert à partager une fonctionnalité commune entre plusieurs cas d'utilisation
- Elle peut également être employée pour structurer un cas d'utilisation en décrivant ses sous-fonctions

# Les relations entre les cas d'utilisation

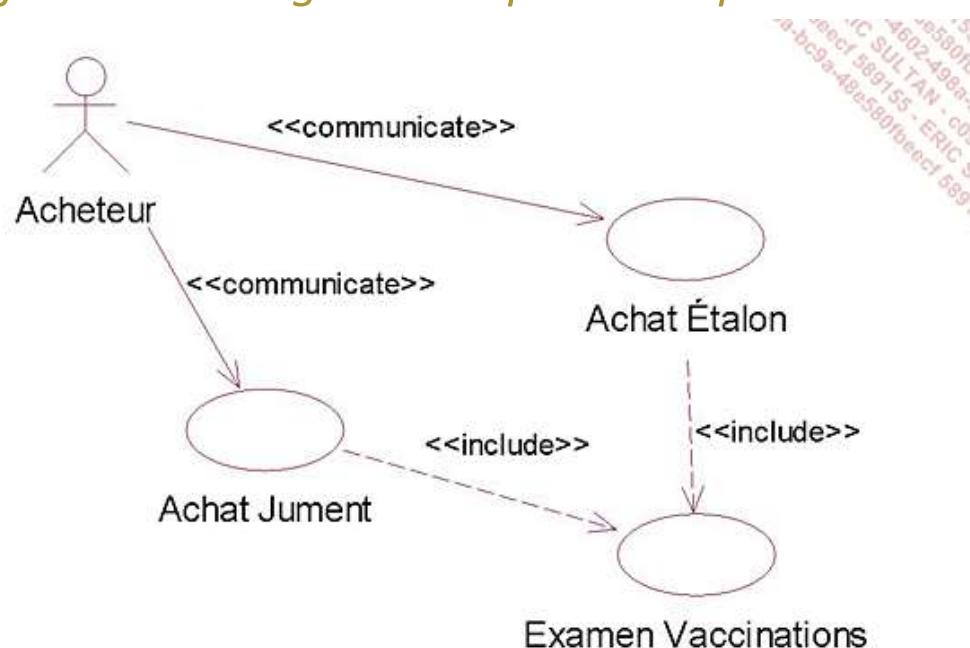
- Dans le diagramme des cas d'utilisation, cette relation est représentée par une flèche pointillée munie du stéréotype «include»
- Exemple :
  - *Lors de l'achat d'un étalon, un acheteur va vérifier ses vaccinations. Par conséquent, le cas d'utilisation d'achat d'un étalon inclut cette vérification*



# Les relations entre les cas d'utilisation

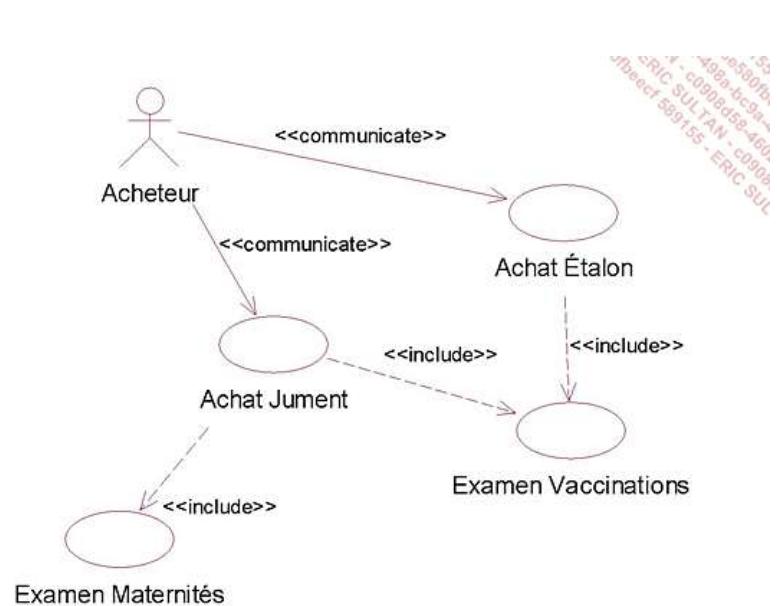
## □ Exemple :

- *La mise en commun du cas d'utilisation d'examen des vaccinations ;*
- *ce cas de sous-fonction est également pertinent pour l'achat d'une jument*



# Les relations entre les cas d'utilisation

- *L'inclusion peut également être employée pour décomposer l'intérieur d'un cas d'utilisation sans que le cas inclus soit partagé*
- *Exemple :*
  - *L'examen des maternités d'une jument n'est pas partagé mais sa présence illustre bien que cet examen fait partie des points étudiés lors de l'achat d'une jument*



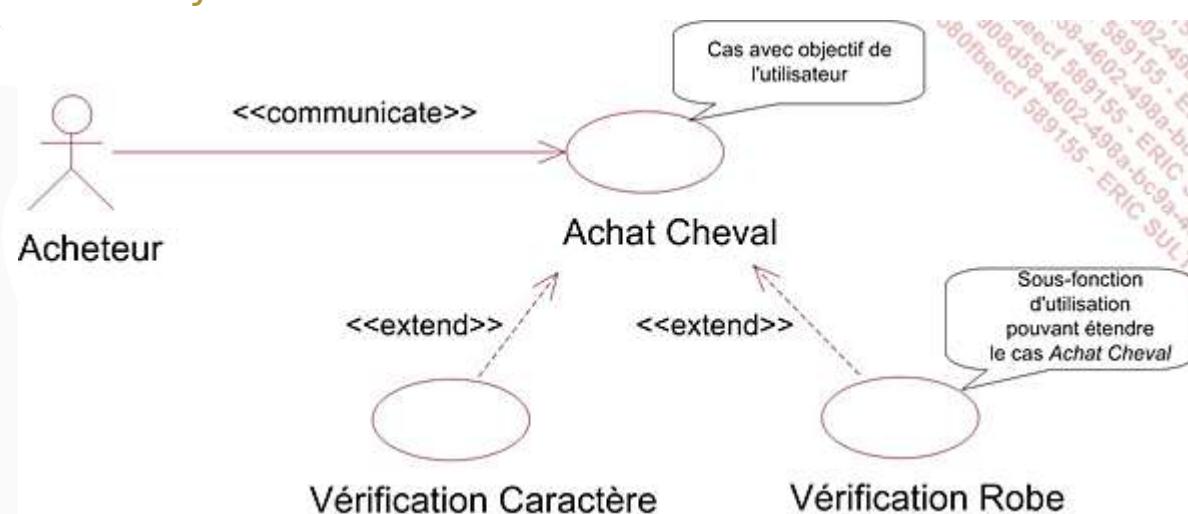
# Les relations entre les cas d'utilisation

## □ La relation d'extension :

- La relation d'extension enrichit un cas d'utilisation par un cas d'utilisation de sous-fonction
- Cet enrichissement est analogue à celui de la relation d'inclusion mais il est optionnel
- L'application de chaque extension est décidée lors du déroulement d'un scénario
- Par conséquent, le cas d'utilisation de base peut être employé sans être étendu
- Comme pour l'inclusion, l'extension sert à structurer un cas d'utilisation ou à partager un cas d'utilisation de sous-fonction

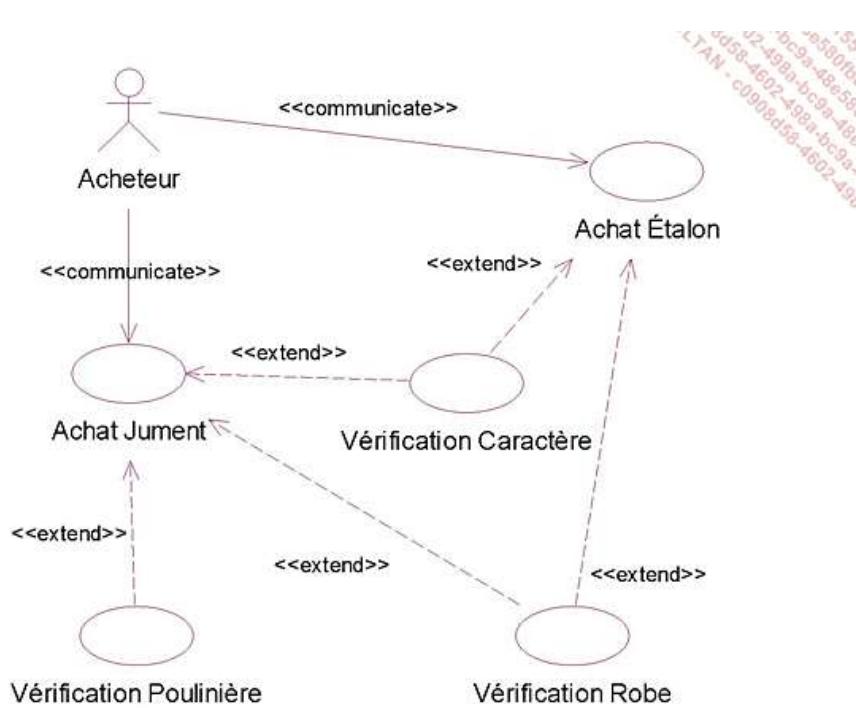
# Les relations entre les cas d'utilisation

- Dans le diagramme des cas d'utilisation, cette relation est représentée par une flèche pointillée munie du stéréotype «extend»
- Exemple :
  - *Lors de l'achat d'un cheval, un acheteur peut vérifier son caractère ou sa robe.*
  - *Par conséquent, le cas d'utilisation d'achat d'un cheval peut être étendu par l'une de ses vérifications*



# Les relations entre les cas d'utilisation

- Exemple :
  - Prenons le cas où l'achat d'un étalon est modélisé séparément de celui d'une jument
  - Sa capacité à donner naissance peut être vérifiée de façon optionnelle
  - D'autre part, les cas d'utilisation de la vérification du caractère et de la vérification de la robe sont partagés



# La spécialisation et la généralisation des cas d'utilisation

- Il est également possible de spécialiser un cas d'utilisation en un autre (cf : Concepts de l'approche par objets)
- Comme pour les classes, le sous-cas hérite du comportement du sur-cas d'utilisation
- Un sous-cas d'utilisation hérite également des relations de communication, d'inclusion et d'extension du sur-cas
- Souvent, le sur-cas d'utilisation est abstrait, c'est-à-dire qu'il correspond à un comportement partiel complété dans les sous-cas d'utilisation

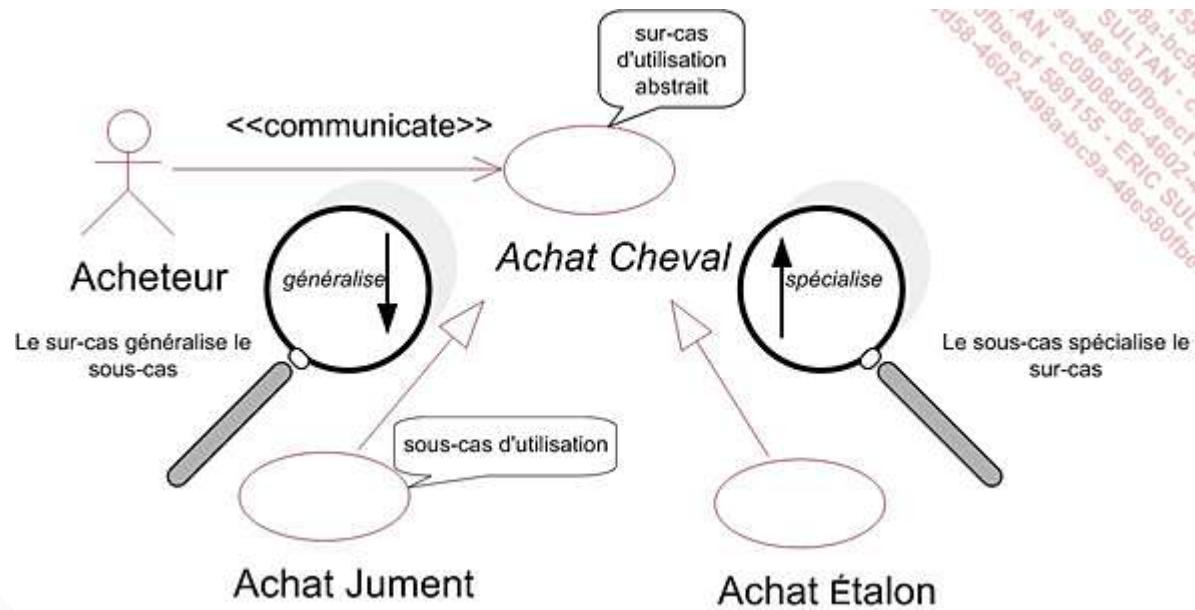
# La spécialisation et la généralisation des cas d'utilisation

- Un sous-cas d'utilisation a le même niveau que son sur-cas :
  - Si le sur-cas est un cas avec objectif utilisateur, il en va de même pour le sous-cas
  - Si le sur-cas est un cas de sous-fonction, le sous-cas est lui aussi une sous-fonction
- Dans le diagramme des cas d'utilisation, la relation de spécialisation est représentée par une flèche pleine de spécialisation identique à celle qui relie les sous-classes aux superclasses
- Le nom d'un cas d'utilisation abstrait est écrit en italique (ou accompagné du stéréotype «abstract»)

# La spécialisation et la généralisation des cas d'utilisation

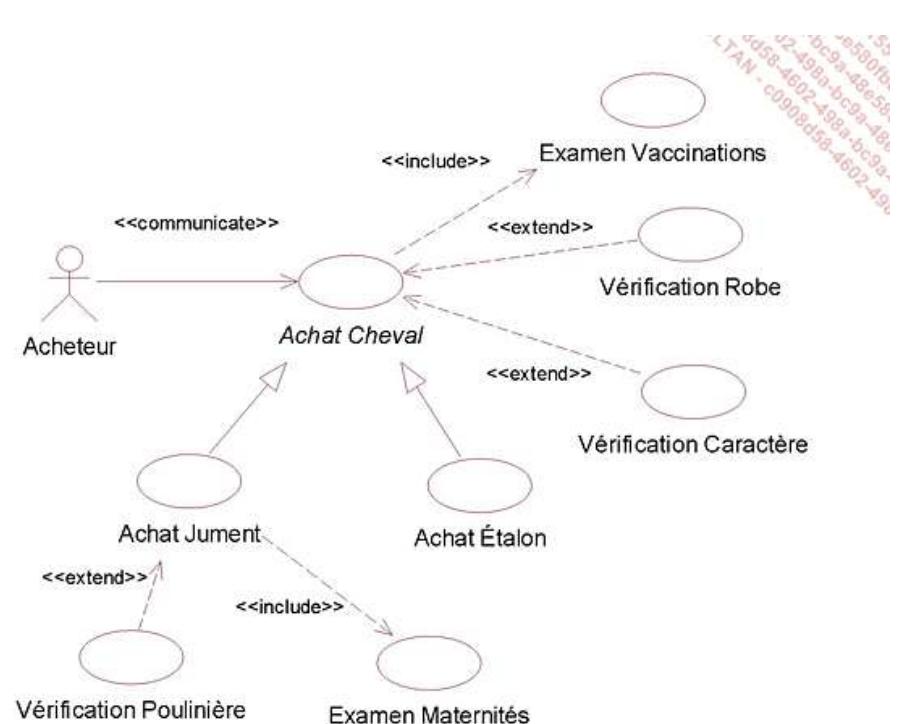
## □ Exemple :

- *Le cas d'utilisation d'achat d'un cheval est spécialisé en deux sous-cas :*
  - *l'achat d'une jument ou d'un étalon*
- *Ce cas est un cas abstrait et son nom apparaît en italiques*



# La spécialisation et la généralisation des cas d'utilisation

- Exemple :
  - *Les relations d'extension concernant les différentes inclusions et extensions de vérification peuvent être factorisées au niveau du cas abstrait*
  - *Elles sont alors héritées dans les sous-cas*



# La représentation textuelle des cas d'utilisation

- La représentation textuelle des cas d'utilisation n'est pas spécifiée dans UML
- Cette représentation sous forme textuelle des cas d'utilisation donne une description de leurs comportements, de leurs actions et réactions

# La représentation textuelle des cas d'utilisation

- Le contenu de cette représentation textuelle est la suivante :
  - 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 utilisateur ;
  - ou 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.

# La représentation textuelle des cas d'utilisation

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 utilisateur 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

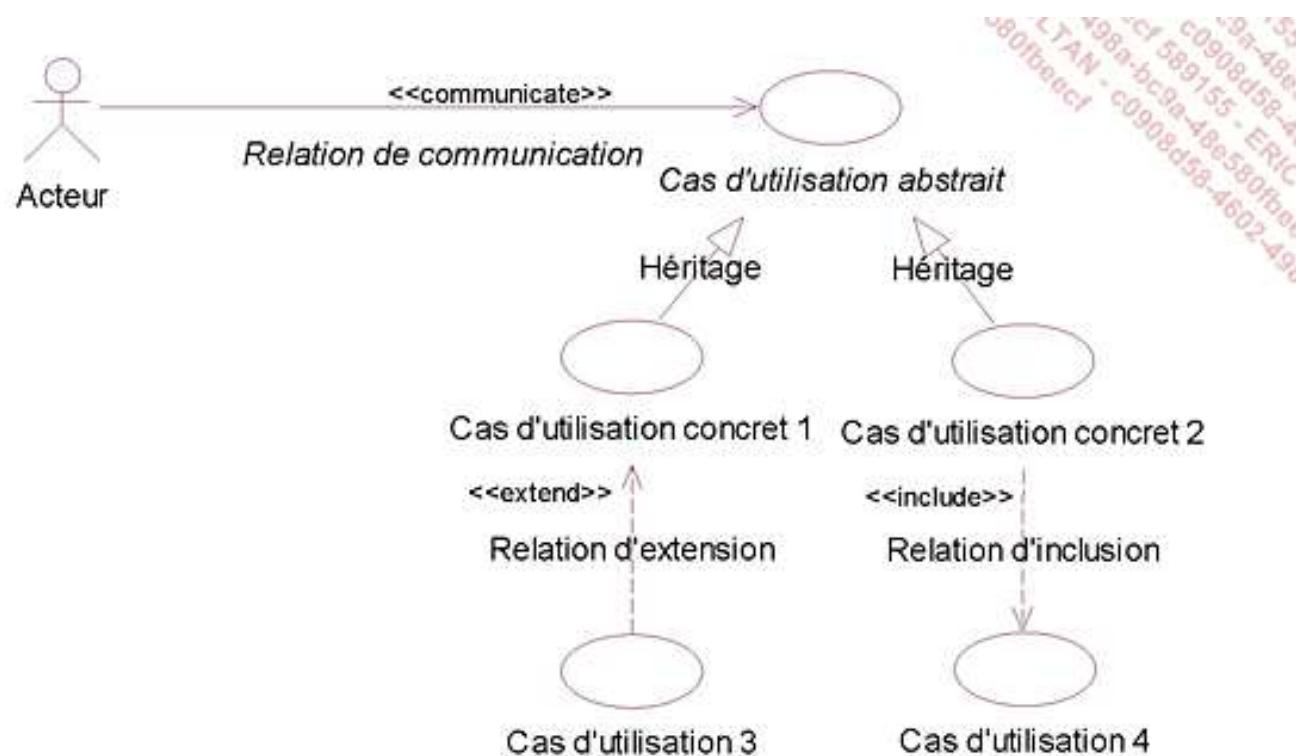
# La représentation textuelle des cas d'utilisation

Cas d'utilisation	Nom du cas d'utilisation
Acteur primaire	Acheteur
Système	Élevage de chevaux
Intervenants	Acheteur, Haras nationaux
Niveau	Objectif utilisateur
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

# Conclusion

- 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.
- Les cas d'utilisation offrent une technique de représentation qui convient au dialogue avec l'utilisateur car son formalisme reste proche du langage naturel
- Il est conseillé d'y adjoindre un lexique pour éviter les risques de confusion

# Diagramme de cas d'utilisation



# **LA MODÉLISATION DE LA DYNAMIQUE**

# Introduction

- L' objectif est de vous faire découvrir comment UML représente les interactions entre les objets
- UML propose deux diagrammes pour répondre à ce besoin de représentation des interactions entre objets :
  - Le diagramme de séquence se focalise sur les aspects temporels
  - Le diagramme de communication se focalise sur la représentation spatiale
- Remarque :
  - Le diagramme de communication porte ce nom depuis UML 2. En UML 1, il s'appelait diagramme de collaboration

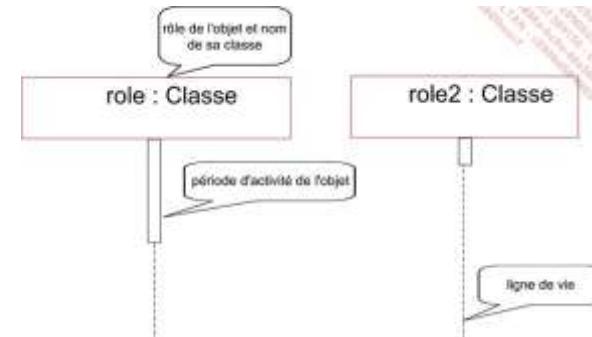
# Le diagramme de séquence

- Le diagramme de séquence décrit la dynamique du système
- Le diagramme de séquence décrit les interactions entre un groupe d'objets en montrant, de façon séquentielle, les envois de message qui interviennent entre les objets
- Le diagramme peut également montrer les flux de données échangées lors des envois de message

# Le diagramme de séquence

## □ La ligne de vie d'un objet :

- Le diagramme de séquence fait entrer en action les instances des classes
- Les acteurs rencontrés dans le diagramme des cas d'utilisation peuvent également être représentés, ils sont considérés comme des instances des classes
- À chaque instance est associée une ligne de vie qui montre ses actions et réactions, ainsi que les périodes pendant lesquelles elle est active, c'est-à-dire où elle exécute l'une de ses méthodes



# Le diagramme de séquence

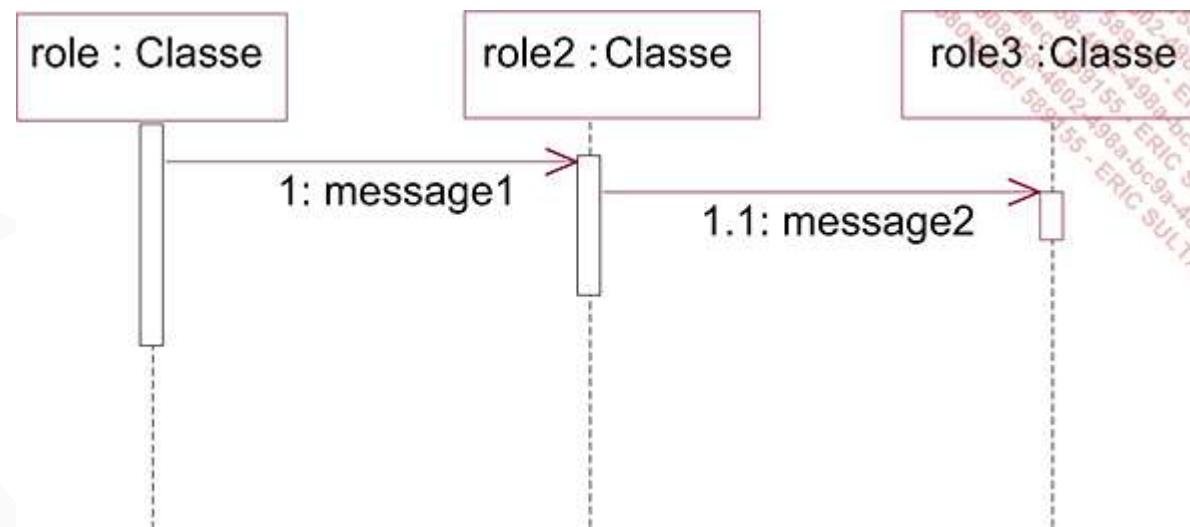
## □ L'envoie de message :

- Les envois de message sont représentés par des flèches horizontales reliant la ligne de vie de l'objet émetteur à la ligne de vie de l'objet destinataire
- L'objet de gauche envoie un message à l'objet de droite. Ce message donne lieu à l'exécution de la méthode *message* de l'objet de droite, ce qui provoque son activation



# Le diagramme de séquence

- Les messages sont numérotés séquentiellement, à partir de un
- Si un message est envoyé alors que le traitement du précédent n'est pas terminé, il est possible d'utiliser une numération composée où l'envoi du message 2 intervient pendant l'exécution du message 1



# Le diagramme de séquence

- La transmission d'information est également possible
- Elle est représentée par des paramètres transmis avec le message



# Le diagramme de séquence

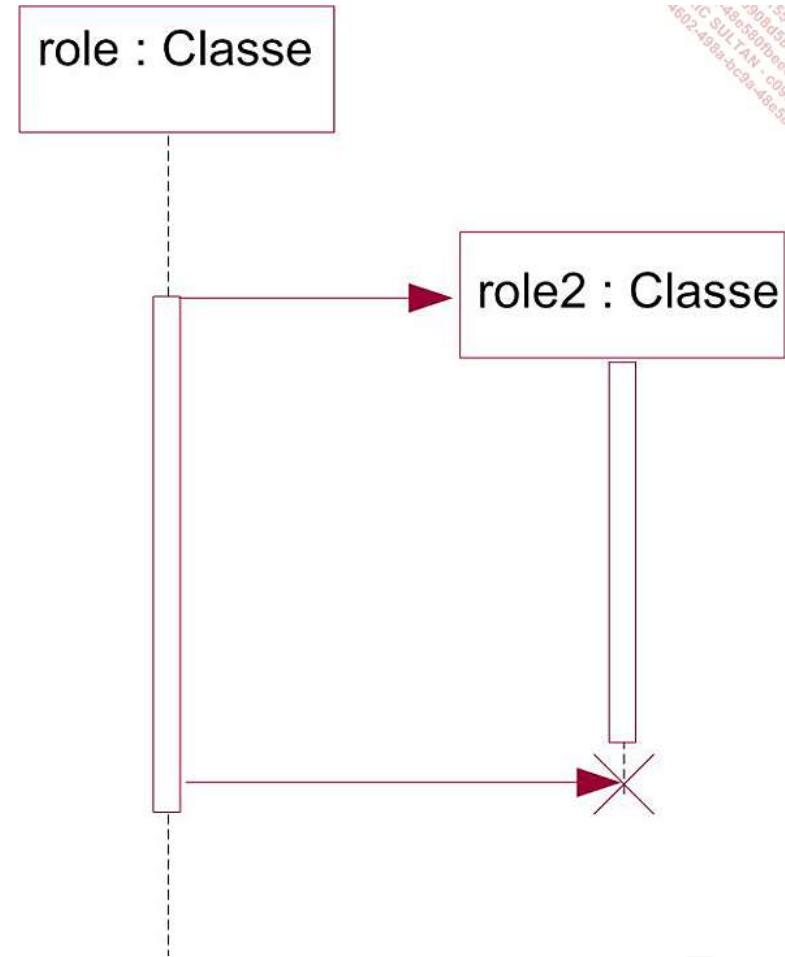
## □ Il existe différents types d'envois de message :

- Le message synchrone est le plus fréquemment utilisé; son emploi signifie que l'expéditeur du message attend que l'activation de la méthode invoquée chez le destinataire soit terminée avant de continuer son activité
- Dans le cas du message asynchrone, l'expéditeur n'attend pas la fin de l'activation de la méthode invoquée chez le destinataire
- Le message de retour de l'invocation d'une méthode n'est pas systématique, toutes les méthodes ne retournant pas un résultat



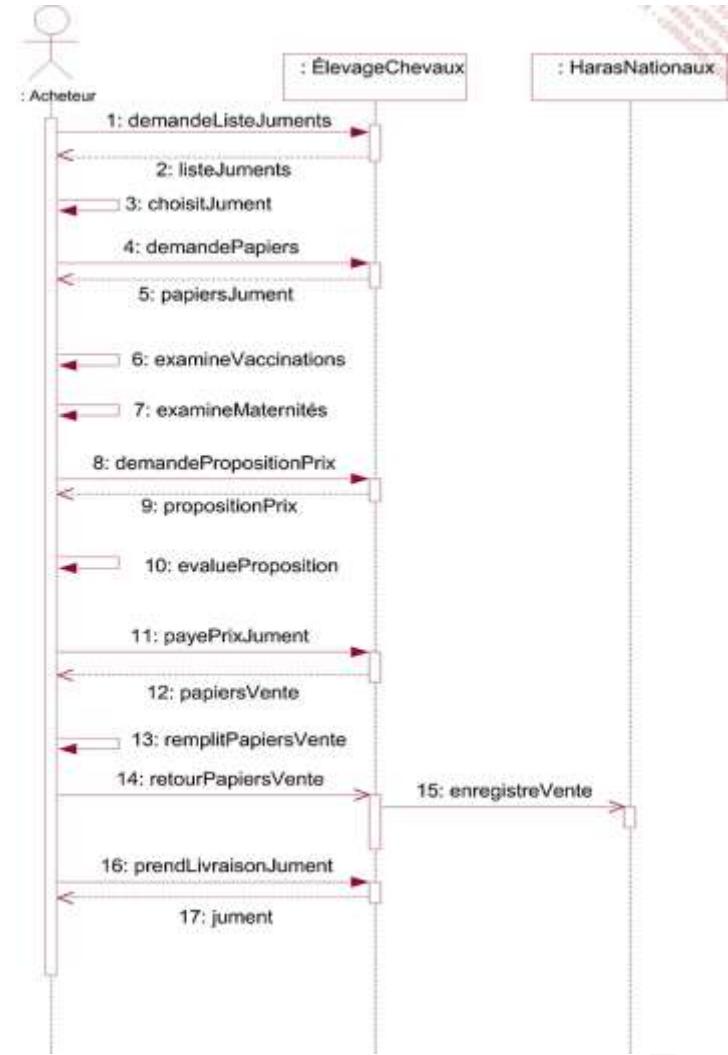
# Le diagramme de séquence

- La création d'objet est représentée par un message spécifique qui donne lieu au début de la ligne de vie du nouvel objet
- La destruction d'objet est un message envoyé à un objet existant et qui donne lieu à la fin de sa ligne de vie. Il est représenté par une croix



# Le diagramme de séquence

- Exemple :
  - Scénario d'achat d'une jument



# Les cadres d'interaction

- Un cadre d'interaction est une partie du diagramme de séquence associé à une étiquette
- Elle contient un opérateur qui en détermine la modalité d'exécution
- Les principales modalités sont le branchement conditionnel et la boucle

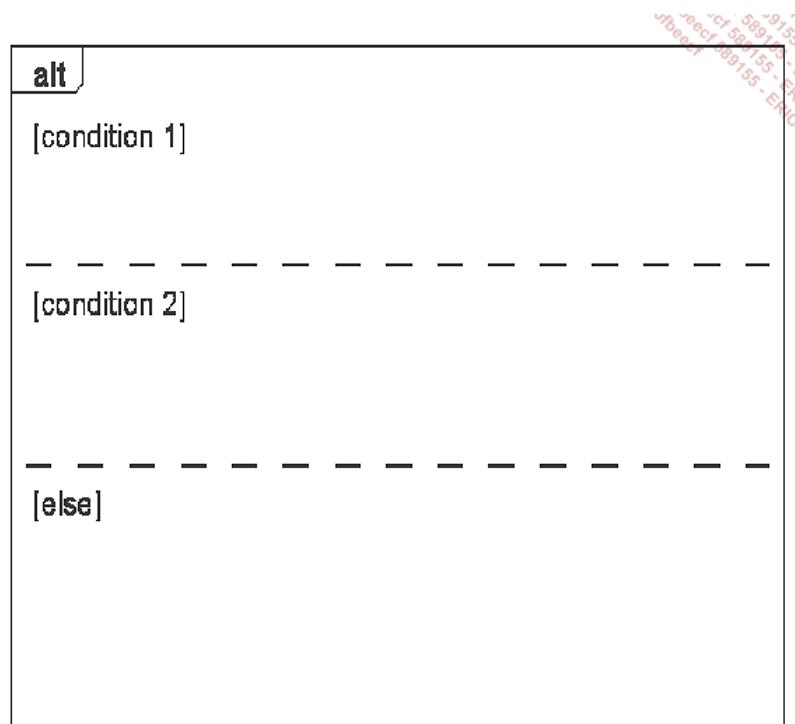
# Les cadres d'interaction

- L'alternative s'obtient en utilisant l'opérateur opt suivi d'une condition de test. Si la condition est vérifiée, le contenu du cadre est exécuté



# Les cadres d'interaction

- Il existe un autre opérateur pour l'alternative. Nommé *alt*, il est suivi de plusieurs conditions de test puis du mot clé *else*



# Les cadres d'interaction

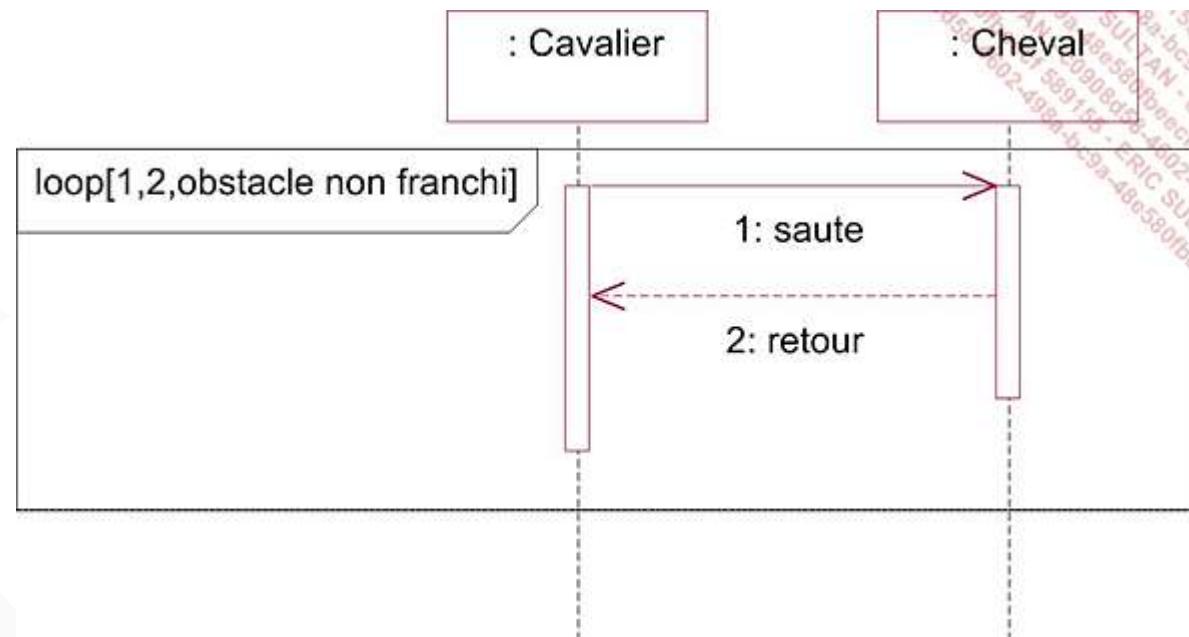
- La boucle est réalisée par l'opérateur *loop* suivi des paramètres *min*, *max* et d'une condition de test

**loop[min,max,condition]**

# Les cadres d'interaction

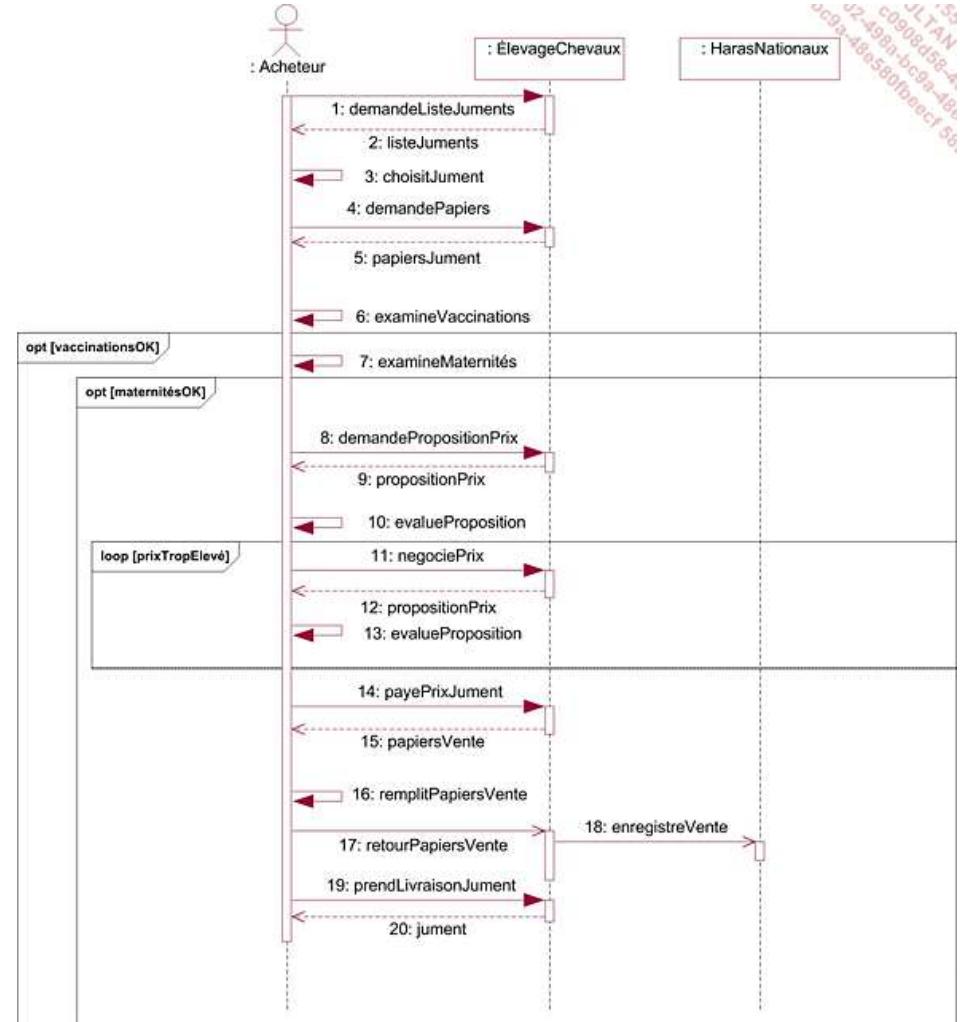
## □ Exemple :

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



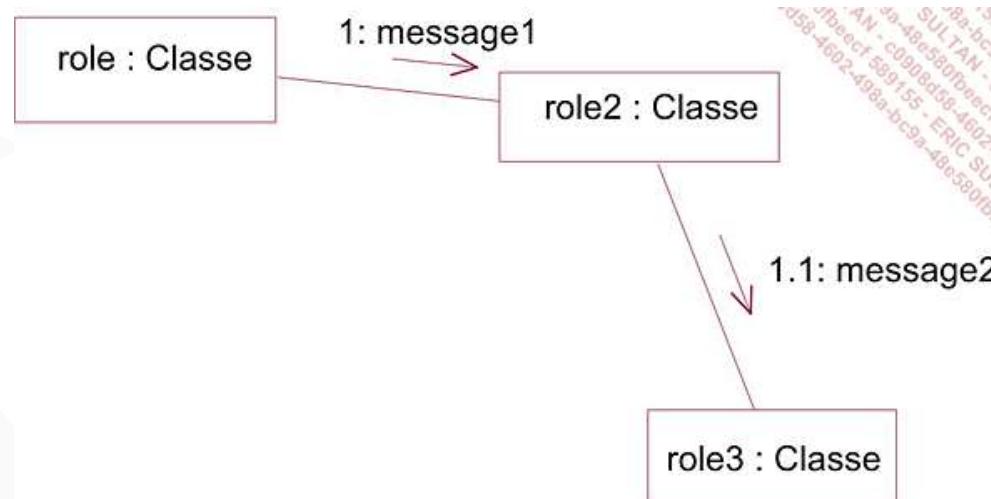
# Les cadres d'interaction

- Exemple :
  - Cas d'utilisation de l'achat d'une jument



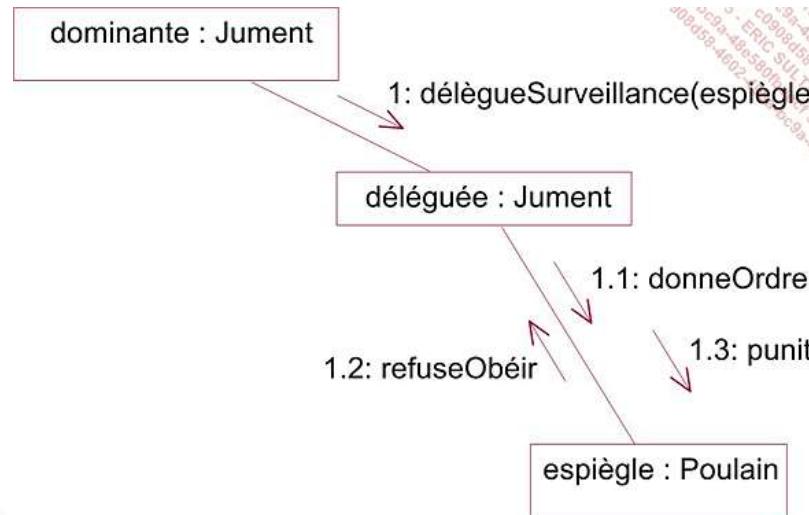
# Le diagramme de communication

- Le diagramme de communication est une alternative au diagramme de séquence
- Il se focalise sur une représentation spatiale des objets
- Il n'est pas associé à une ligne de vie mais relié graphiquement aux objets avec lesquels il interagit



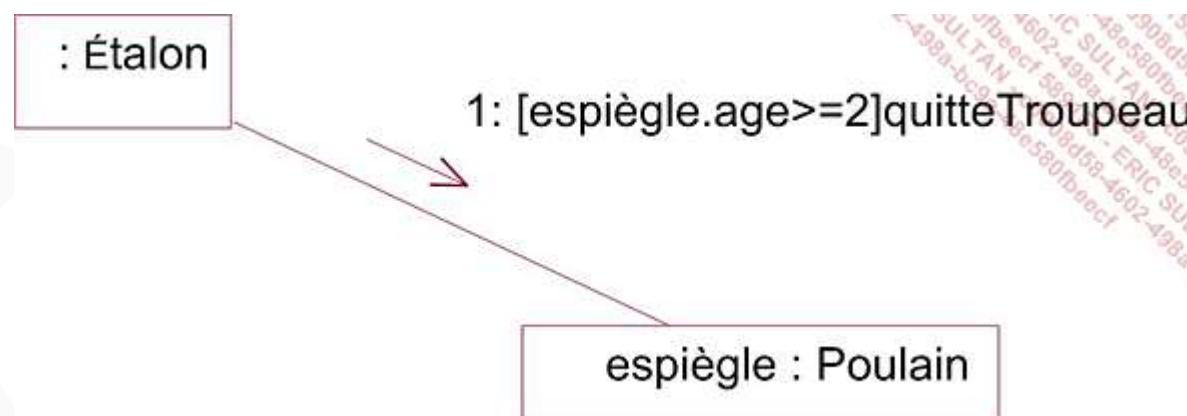
# Le diagramme de communication

- Il est également possible de transmettre des informations lors de l'envoi
- Exemple :
  - *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*



# Le diagramme de communication

- UML propose des mécanismes de test et de boucle au niveau de l'envoi des messages
- Exemple :
  - *Quand un poulain atteint l'âge de deux ans, il est chassé du troupeau par l'étalon*

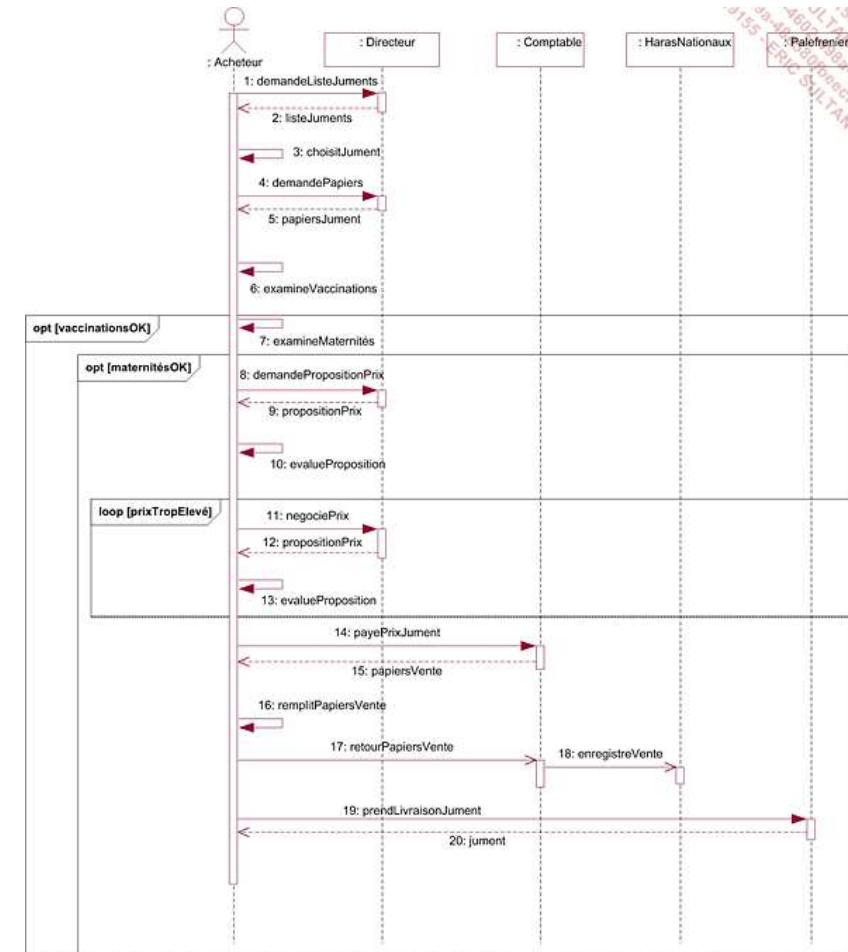


# Découvrir les objets du système

- Pour découvrir les objets du système à partir d'un cas d'utilisation, une première phase consiste à se demander à quels objets du système sont destinés les messages provenant de l'extérieur
- Leur détermination constitue une première étape de décomposition du système en objets

# Découvrir les objets du système

- Exemple :
  - *Dans l'exemple du cas d'utilisation d'achat d'une jument, les interactions entre l'acheteur et l'élevage de chevaux peuvent être décomposées en interactions entre d'une part l'acheteur et d'autre part le directeur, le comptable ou le palefrenier de l'élevage*

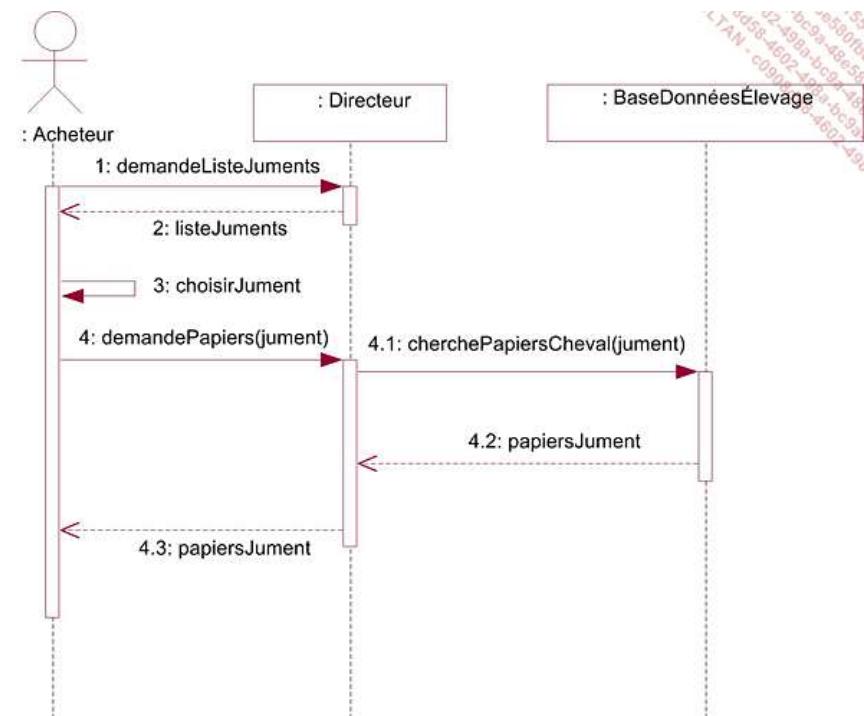


# Découvrir les objets du système

- Dans un second temps, les messages reçus par les objets du système sont décomposés progressivement
- On continue ainsi de découvrir au fur et à mesure les objets du système.
- En effet, la décomposition des messages oblige à utiliser de nouveaux objets qui répondent aux fonctionnalités requises
- Le traitement de ces informations implique souvent l'utilisation de nouveaux objets

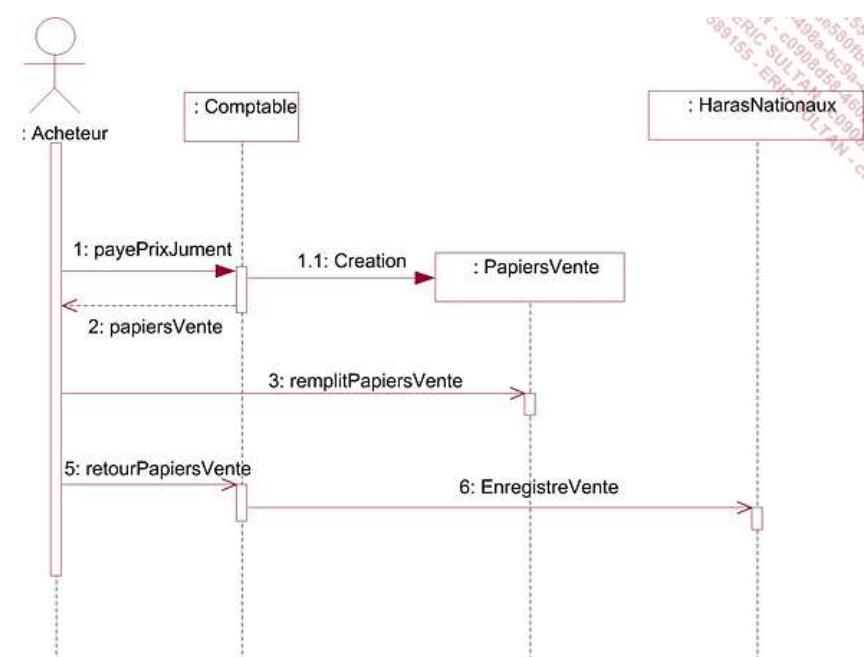
# Découvrir les objets du système

- Exemple :
  - *Lorsque le directeur reçoit la demande des papiers de la jument, il fait appel à la base de données de l'élevage pour les retrouver*



# Découvrir les objets du système

- Exemple :
  - *Lorsque le comptable reçoit le paiement, il crée les papiers de la vente avant de les renvoyer à l'acheteur*
  - *Cette décomposition introduit un nouvel objet du système : ces papiers*



# Conclusion

- 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
- Depuis l'introduction des cadres d'interaction, 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

# **LA MODÉLISATION DES OBJETS**

# Introduction

- Cette modélisation est statique car elle ne décrit pas les interactions ou le cycle de vie des objets
- Les méthodes sont introduites d'un point de vue statique, sans description de leur enchaînement
- Nous découvrirons le diagramme des classes
- De tous les diagrammes UML, il est le seul obligatoire lors d'une telle modélisation

# Introduction

- Nous étudierons comment le langage OCL (*Object Constraint Language* ou langage de contraintes *objet*) peut étendre le diagramme des classes pour exprimer de façon plus riche les contraintes
- Enfin, le diagramme des objets nous montrera comment illustrer la modélisation réalisée dans le diagramme des classes
- L'emploi d'OCL ou du diagramme des objets est optionnel

# Découvrir les objets du système par décomposition

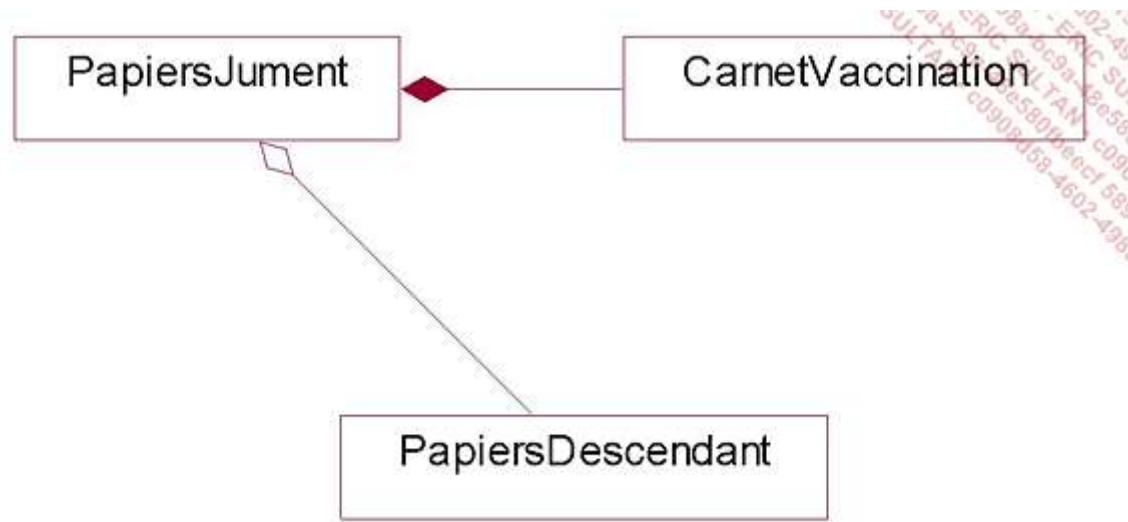
- Nous avons vu que la décomposition des messages fait apparaître les objets du système, car elle conduit à des messages plus fins dont il convient de rechercher le destinataire
- Une autre approche possible est la décomposition de l'information contenue dans un objet
- Souvent, cette information est trop complexe pour n'être représentée que par la structure d'un seul objet
- Elle doit parfois être également répartie entre plusieurs objets

# Découvrir les objets du système par décomposition

## □ Exemple :

- *Dans l'exemple de 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*

# Découvrir les objets du système par décomposition

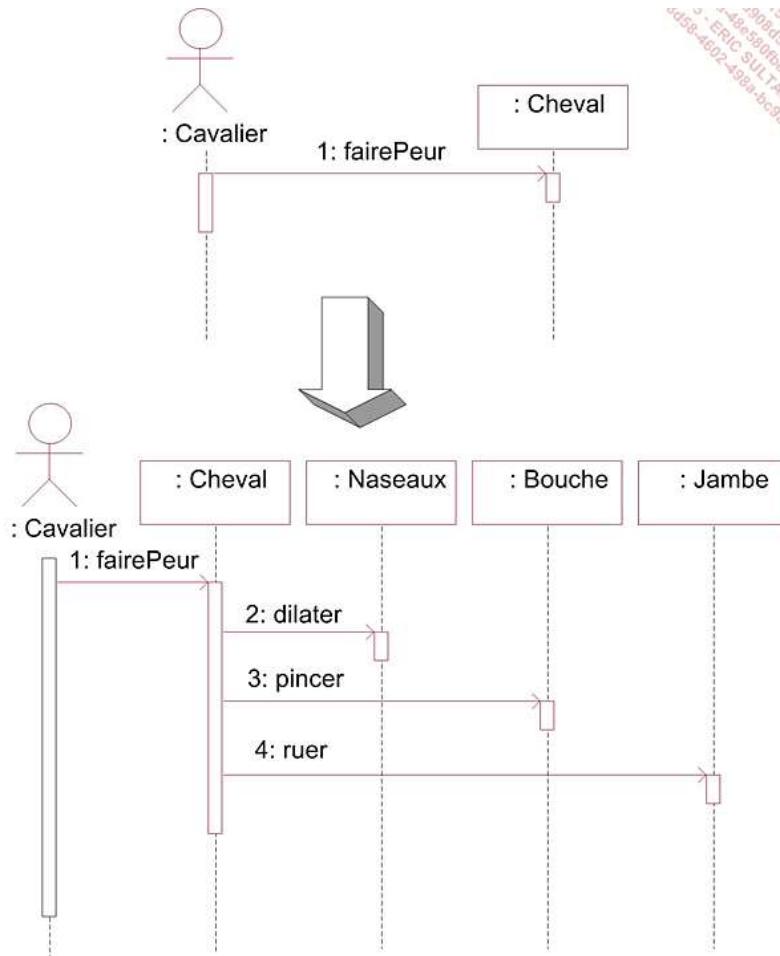


# Découvrir les objets du système par décomposition

## □ Exemple :

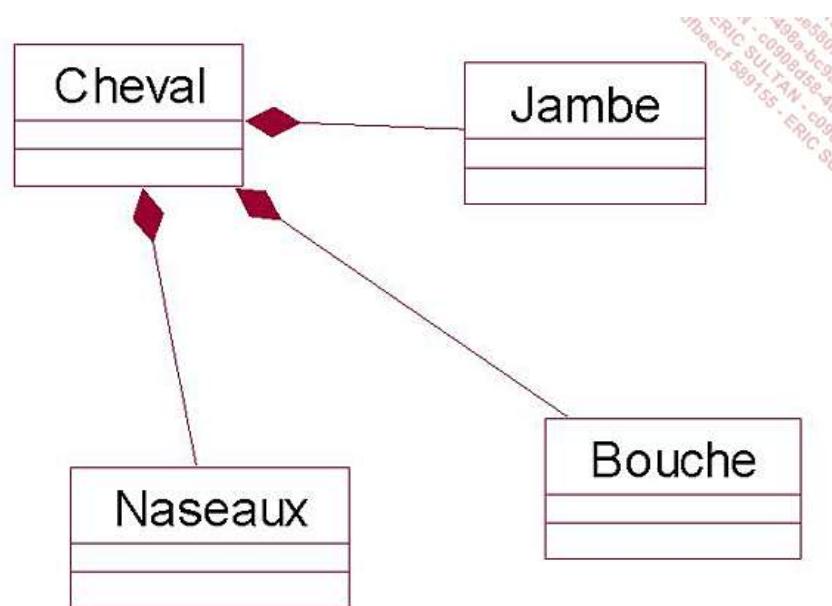
- *La décomposition d'un cheval pour faire apparaître ses différents organes peut se faire soit par la décomposition d'un diagramme de séquence, soit par la décomposition guidée par les données*
- *La décomposition par le diagramme de séquence consiste à analyser différents envois de message : faire peur, courir, manger, dormir*
- *Ces derniers feront apparaître progressivement les différents organes du cheval*
- *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.*

# Découvrir les objets du système par décomposition



# Découvrir les objets du système par décomposition

- La décomposition guidée par les données consiste à étudier directement les différents organes d'un cheval et à les prendre en compte dans le diagramme des classes



# La représentation des classes

- Cette représentation est constituée de trois parties :
  - La première partie contient le nom de la classe
  - La deuxième partie contient les attributs. Ceux-ci contiennent l'information portée par un objet. L'ensemble des attributs forme la structure de l'objet
  - La troisième partie contient les méthodes. Celles-ci correspondent aux services offerts par l'objet. Elles peuvent modifier la valeur des attributs. L'ensemble des méthodes forme le comportement de l'objet.

<b>NomClasse</b>
nomAttribut1
nomAttribut2
nomAttribut3
nomMéthode1()
nomMéthode2()

# La représentation des classes

□ Exemple :

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

# La représentation des classes

## □ L'encapsulation :

- Certains attributs et méthodes ne sont pas exposés à l'extérieur de l'objet
- Ils sont encapsulés et appelés attributs et méthodes privés de l'objet

## □ UML introduit trois possibilités d'encapsulation :

- L'attribut 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 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)

# La représentation des classes

- L'encapsulation est représentée par un signe plus, un signe moins, un dièse, ou un tilde précédant le nom de l'attribut
- Le tableau suivant détaille la signification de ces signes :

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

# La représentation des classes

## □ La notion de type :

- Nous appelons ici variable tout attribut, paramètre et valeur de retour d'une méthode
- D'une façon générale, nous appelons variable tout élément pouvant prendre une valeur
- Le type est une contrainte appliquée à une variable
- Il consiste à fixer l'ensemble des valeurs possibles que peut prendre cette variable
- Cet ensemble peut être une classe, auquel cas la variable doit contenir une référence vers une instance de cette classe
- Il peut être standard comme l'ensemble des entiers, des chaînes de caractères, des booléens ou des réels

# La représentation des classes

## □ Les types standard sont désignés ainsi :

- Integer pour le type des entiers
- String pour le type des chaînes de caractères
- Boolean pour le type des booléens
- Real pour le type des réels.

## □ Exemple :

- *1 ou 3 ou 10 sont des exemples de valeurs d'entier.*
- *"Cheval" est un exemple de chaîne de caractères où les guillemets ont été choisis comme séparateurs.*
- *False et True sont les deux seules valeurs possibles du type Boolean.*
- *3.1415, où le point a été choisi comme séparateur des décimales, est un exemple bien connu de nombre réel.*

# La représentation des classes

## □ Exemple :



# La représentation des classes

## □ La signature des méthodes :

- Une méthode d'une classe peut prendre des paramètres et renvoyer un résultat
- Les paramètres sont des valeurs transmises :
  - à l'aller, lors de l'envoi d'un message appelant la méthode
  - ou au retour d'appel de la méthode
- Le résultat est une valeur transmise à l'objet appelant lors du retour d'appel
- L'ensemble constitué du nom de la méthode, des paramètres avec leur nom et leur type ainsi que le type du résultat s'appelle la signature de la méthode.
- Une signature prend la forme suivante :
  - <nomMéthode> (<direction><nomParamètre> : <type>, ...) :<typeRésultat>

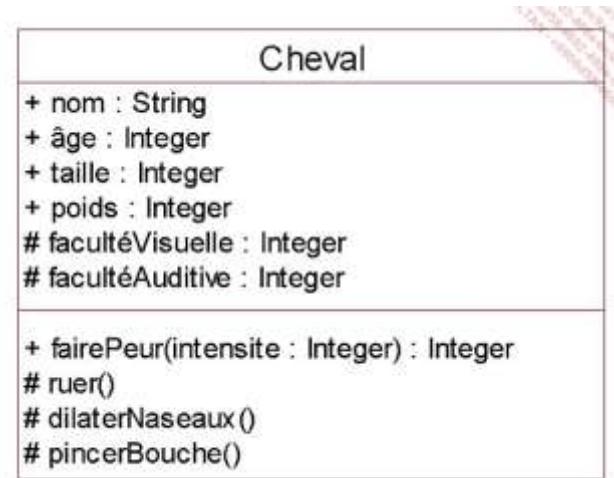
# La représentation des classes

- Rappelons que le nombre de paramètres peut être nul et que le type du résultat est optionnel
- Devant le nom du paramètre, il est possible d'indiquer par un mot clé la direction dans laquelle celui-ci est transmis
- Les trois mots clés possibles sont :
  - 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
- Si aucun mot clé n'est précisé, la valeur du paramètre n'est transmise qu'à l'appel

# La représentation des classes

## □ 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.*



# La représentation des classes

## □ La forme complète de représentation des classes

- La représentation complète des classes fait apparaître les attributs avec leur caractéristique d'encapsulation, leur type et les méthodes avec leur signature complète
- Il est également possible de donner des valeurs par défaut aux attributs et aux paramètres d'une méthode
- La valeur par défaut d'un attribut est celle qui lui est attribuée dès la création d'un nouvel objet
- La valeur par défaut d'un paramètre est utilisée lorsque l'appelant d'une méthode ne fournit pas explicitement la valeur de ce paramètre au moment de l'appel

NomClasse
+ nomAttribut1 : typeAttribut1 = valeurDéfaut # nomAttribut2 : typeAttribut2 = valeurDéfaut + nomAttribut3 : typeAttribut3 = valeurDéfaut
+ nomMéthode1(param1 : typeParam1 = valeurDéfaut, param2 : typeParam2) : typeRetour # nomMéthode2(param : typeParam = valeurDéfaut) : typeRetour

# La représentation des classes

## □ Les attributs et les méthodes de classe

- Chaque instance d'une classe contient une valeur spécifique pour chacun de ses attributs
- Cette valeur n'est donc pas partagée par l'ensemble des instances
- Dans certains cas, il est nécessaire d'utiliser des attributs dont la valeur est commune à l'ensemble des objets d'une classe
- Un tel attribut voit sa valeur partagée au même titre que son nom, son type et sa valeur par défaut
- Un tel attribut est appelé *attribut de classe* car il est lié à la classe
- Un attribut de classe est représenté par un nom souligné
- Il peut être encapsulé et posséder un type
- Il est vivement recommandé de lui donner une valeur par défaut

# La représentation des classes

## □ Exemple :

- *Nous étudions un système qui est une boucherie exclusivement chevaline*
- *Nous 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 55, soit 5,5 %, le taux de TVA des produits alimentaires en France.*

# La représentation des classes

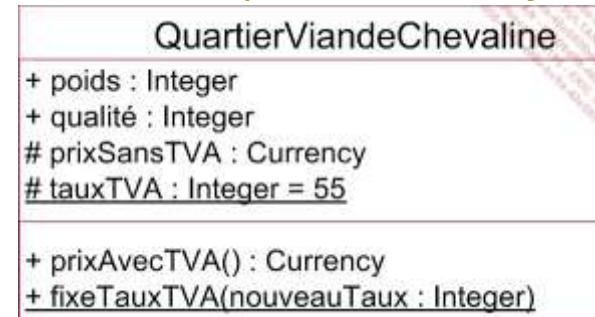
## QuartierViandeChevaline

+ poids : Integer  
+ qualité : Integer  
# prixSansTVA : Currency  
# tauxTVA : Integer = 55

+ prixAvecTVA() : Currency

# La représentation des classes

- Au sein d'une classe, il peut également exister une ou plusieurs méthodes de classe
- Pour appeler une méthode de classe, il faut envoyer un message à la classe elle-même et non à l'une de ses instances.
- Une telle méthode ne manipule que les attributs de classe.
- Exemple :
  - ajoute une méthode de classe à la classe *QuartierViande-Chevaline* qui sert à fixer le taux de TVA.
  - En effet, celui-ci est fixé par la loi et cette dernière peut être modifiée.



# La représentation des classes

## □ Les attributs calculés

- UML introduit la notion d'attribut calculé dont la valeur est donnée par une fonction basée sur la valeur d'autres attributs
- Un tel attribut possède un nom précédé du signe / et il est suivi d'une contrainte donnant le moyen de calculer sa valeur

## □ Exemple :

- *La méthode prixAvecTVA est remplacée par l'attribut calculé /prixAvecTVA*

QuartierViandeChevaline
+ poids : Integer + qualité : Integer # prixSansTVA : Currency # tauxTVA : Integer = 55 /+ prixAvecTVA : Currency {/prixAvecTVA = (1+tauxTVA/1000)*prixSansTVA}

# Les associations entre objets

## □ Les liens entre objets :

- Dans le monde réel, de nombreux objets sont liés entre eux
- Ce lien correspond à une association qui existe entre les objets

## □ 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*

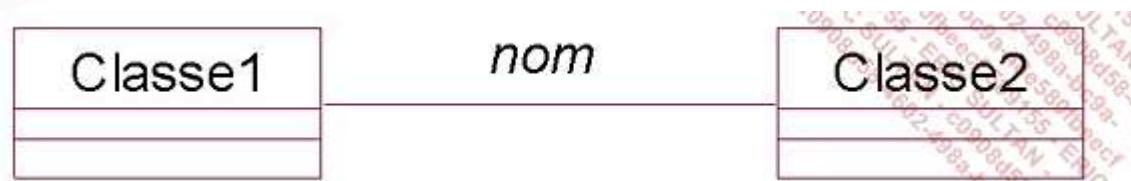
# Les associations entre objets

- En UML, ces liens sont décrits par une association, comme un objet est décrit par une classe. Un lien est une occurrence d'une association
- Par conséquent, une association relie des classes. Chaque occurrence de cette association relie entre elles des instances de ces classes.
- Une association porte un nom. Comme pour une classe, ce nom est significatif des occurrences de l'association.
- 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 entre objets

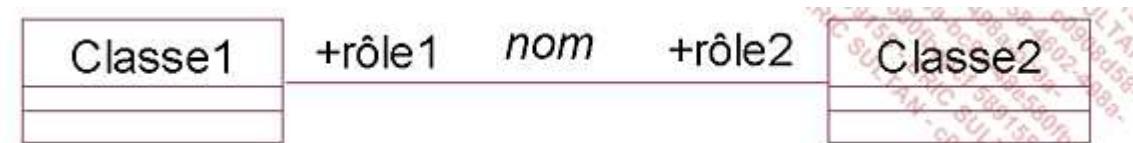
## □ La représentation des associations entre les classes :

- La représentation graphique d'une association binaire consiste en un trait continu entre les deux classes dont elle associe les instances
- Il est possible de faire précéder le nom d'une association du signe < ou de le faire suivre par le signe > pour indiquer le sens de lecture du nom vis-à-vis du nom des classes. Si l'association est située sur un axe vertical, il est possible de faire précéder le nom par ^ ou v.



# Les associations entre objets

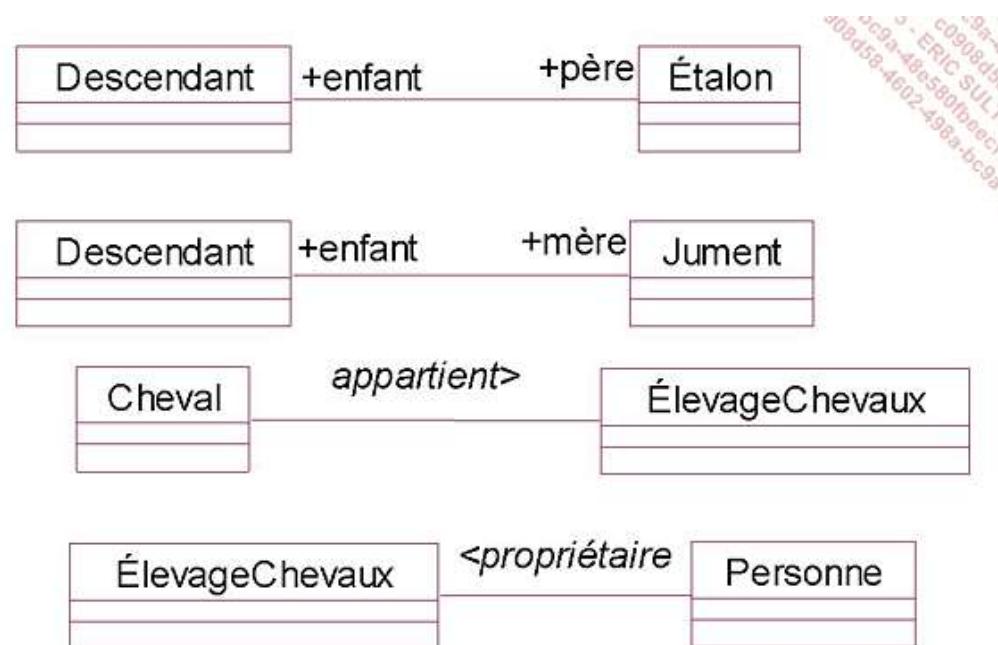
- Chaque extrémité d'une association peut également être nommée
- Ce nom est significatif du rôle que jouent les instances de la classe correspondante dans l'association
- Un rôle a la même nature qu'un attribut dont le type serait la classe située à l'autre extrémité
- Par conséquent, il peut être public ou encapsulé de façon privée, protégée ou pour le paquetage



# Les associations entre objets

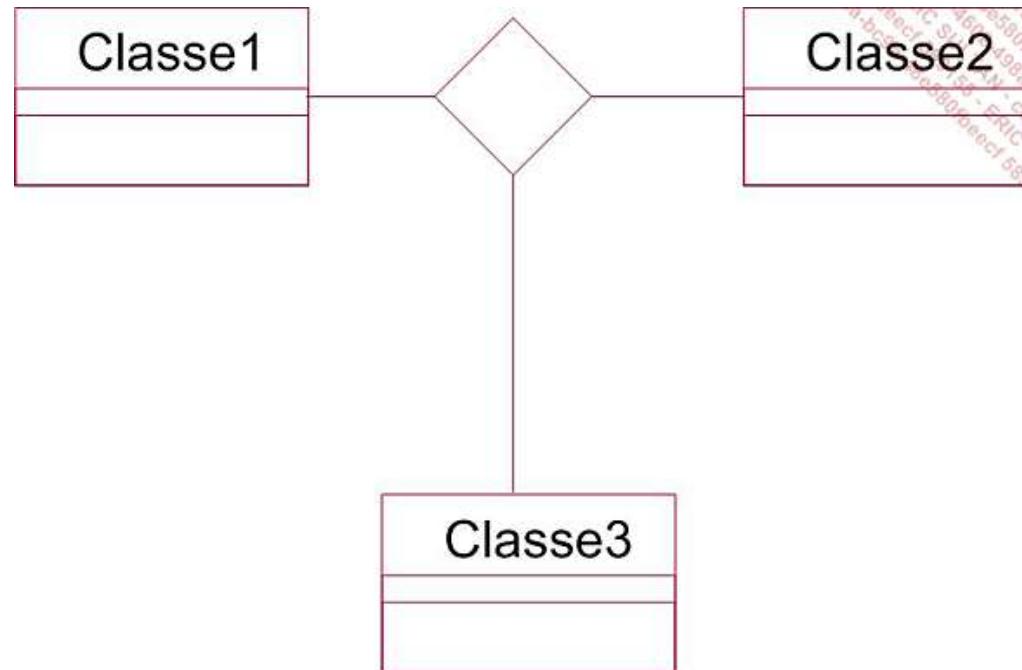
## □ Exemple :

- Pour ces associations, soit le nom de l'association, soit ses rôles ont été indiqués



# Les associations entre objets

- La représentation graphique d'une association ternaire et au-delà consiste en un losange qui relie les différentes classes

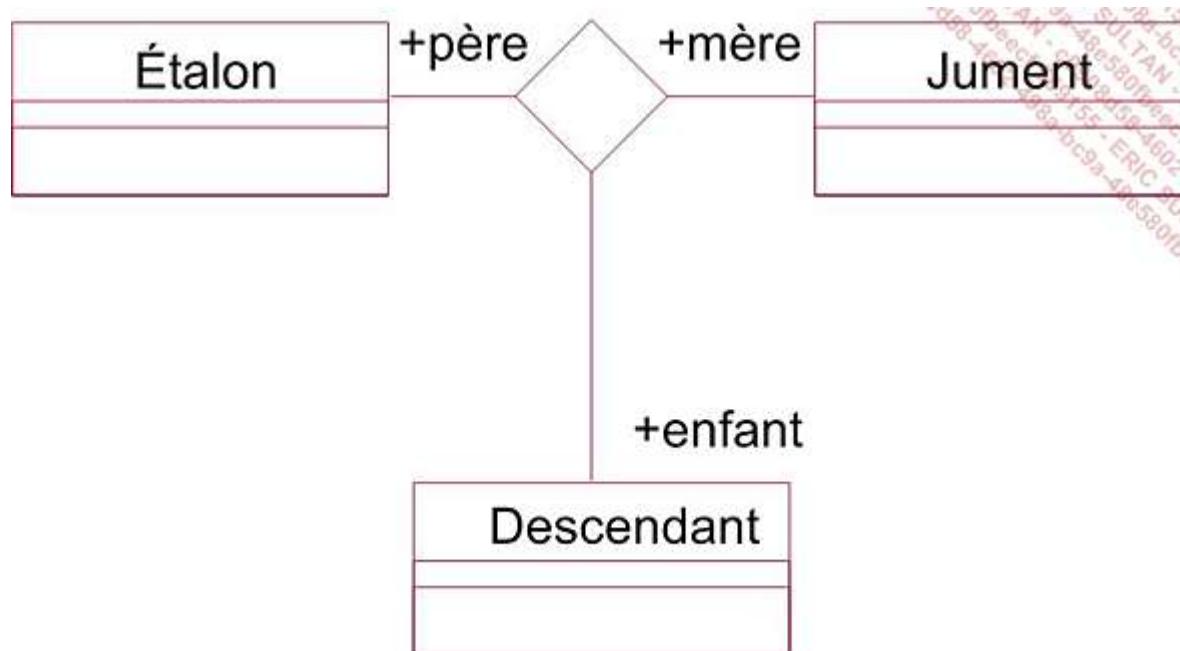


# Les associations entre objets



Exemple :

- *Chacune de ses occurrences constitue un triplet (père, mère, poulain).*



# Les associations entre objets

## □ La cardinalité des associations :

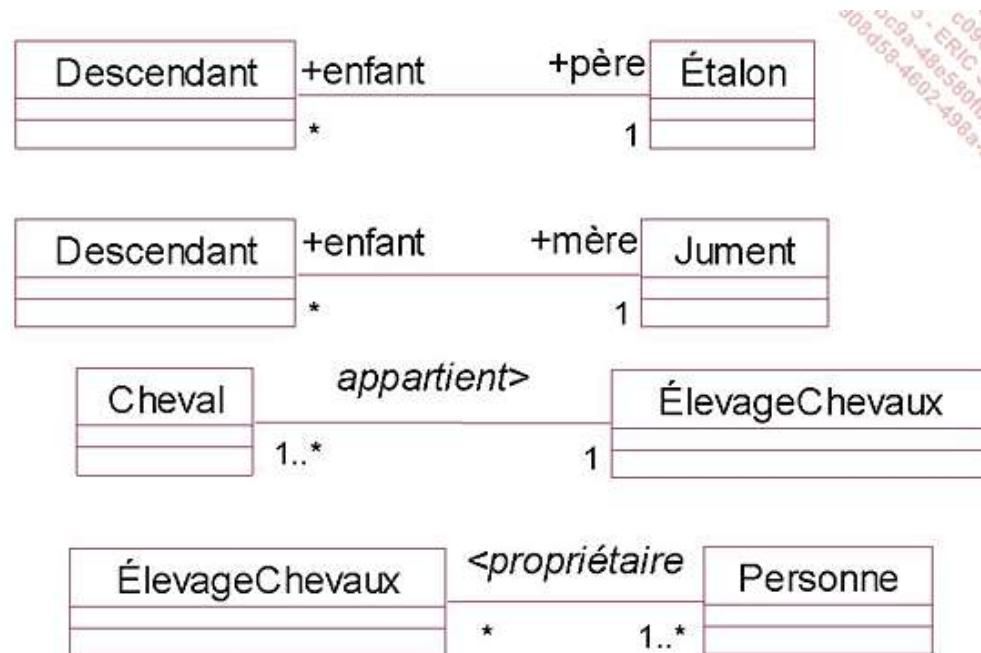
- La cardinalité située à une extrémité d'une association indique à combien d'instances de la classe située à cette extrémité, une instance de la classe située à l'autre extrémité est liée
- Il est possible de spécifier, à une extrémité d'une association, la cardinalité minimale et la cardinalité maximale pour indiquer un intervalle de valeurs auquel doit toujours appartenir la cardinalité

Spécification	Cardinalités
0..1	Zéro ou une fois
1	Une et une seule fois
*	De zéro à plusieurs fois
1..*	De un à plusieurs fois
M..N	Entre M et N fois
N	N fois

# Les associations entre objets

## □ Exemple :

- *Un élevage peut avoir plusieurs copropriétaires et une personne peut être propriétaire de plusieurs élevages.*



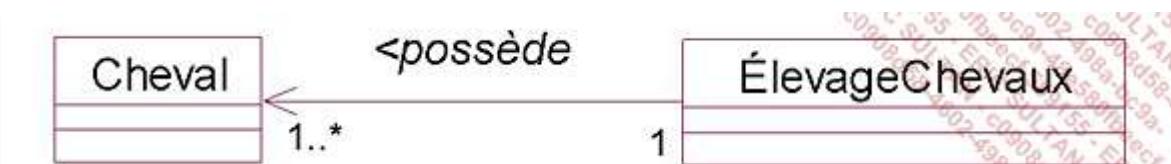
# Les associations entre objets

## □ Navigation :

- Les associations ont par défaut une navigation bidirectionnelle, c'est-à-dire qu'il est possible de déterminer les liens de l'association depuis une instance de chaque classe d'origine
- Une navigation bidirectionnelle est plus complexe à réaliser par les développeurs ; il convient de l'éviter dans la mesure du possible

## □ Exemple :

- *Dans le cadre particulier d'un élevage de chevaux, il est utile de connaître les chevaux que cet élevage possède mais le contraire n'est pas nécessaire*



# Les associations entre objets

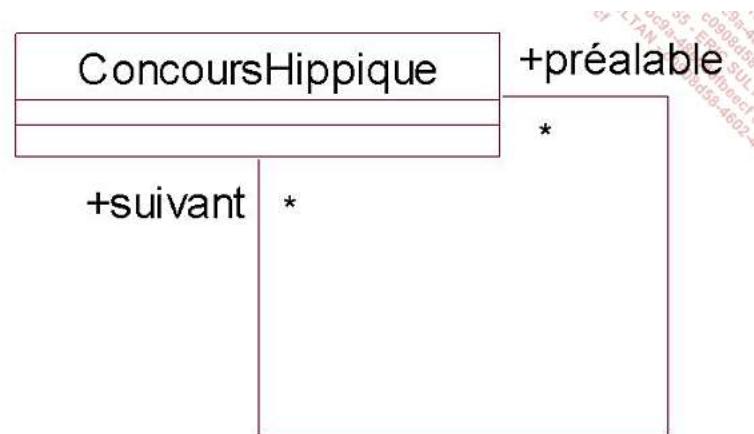
## □ Associer une classe avec elle-même

- La même classe peut se trouver à chaque extrémité d'une association.
- Il s'agit alors d'une association réflexive, reliant entre elles les instances d'une même classe
- Comme nous le verrons dans les exemples, une association réflexive sert principalement à décrire au sein de l'ensemble des instances d'une classe :
  - des groupes d'instances
  - ou une hiérarchie au sein des instances

# Les associations entre objets

## □ Exemple :

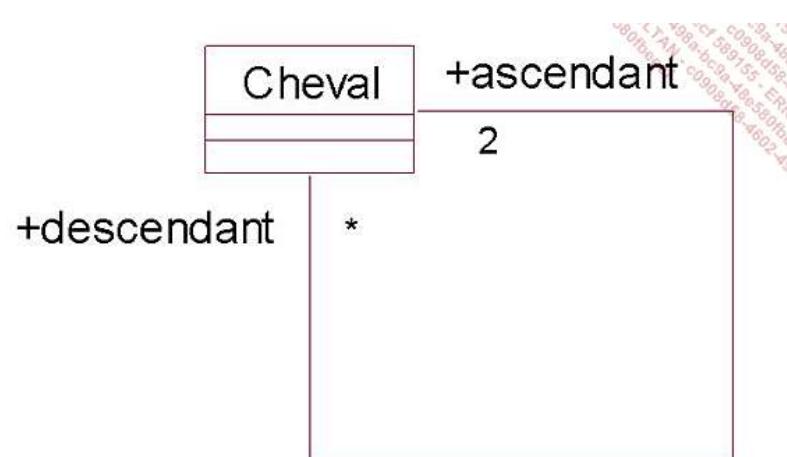
- *Pour pouvoir passer les épreuves de sélection d'un concours hippique international, un cheval doit avoir gagné d'autres concours préalables*
- *Il est donc possible de créer une association entre un concours et ses concours préalables*
- *Cette association crée une hiérarchie au sein des concours*



# Les associations entre objets

## □ Exemple :

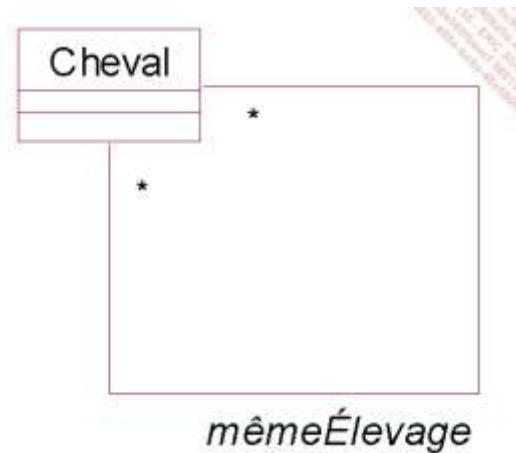
- Illustrons l'association "ascendant/descendant direct" entre les chevaux. Cette association crée une hiérarchie au sein des chevaux
- Pour les ascendants, la cardinalité est 2 car tout cheval a exactement une mère et un père



# Les associations entre objets

## □ Exemple :

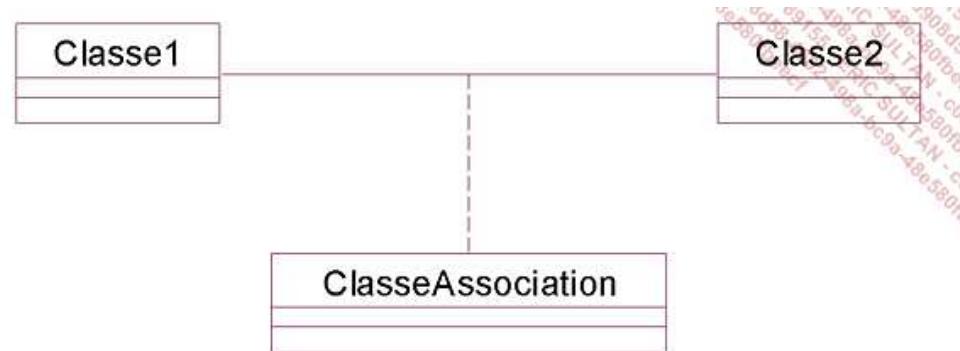
- *Illustrons l'association entre les chevaux qui se trouvent dans le même élevage*
- *Cette association crée des groupes au sein de l'ensemble des instances de la classe Cheval, chaque groupe correspondant à un élevage*



# Les associations entre objets

## □ Les classes-associations :

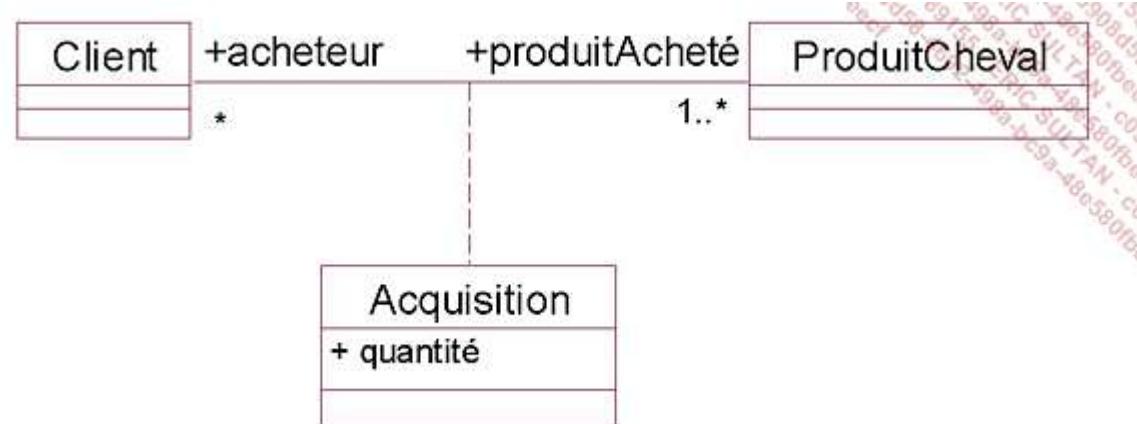
- Les liens entre les instances des classes peuvent porter des informations
- Celles-ci sont spécifiques à chaque lien
- Dans ce cas, l'association qui décrit de tels liens reçoit le statut de classe, dont les instances sont des occurrences de l'association
- Comme toute autre classe, une telle classe peut être dotée d'attributs, d'opérations, et être reliée au travers d'associations à d'autres classes



# Les associations entre objets

## □ Exemple :

- *Quand 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*



# Les associations entre objets

## □ La qualification des associations :

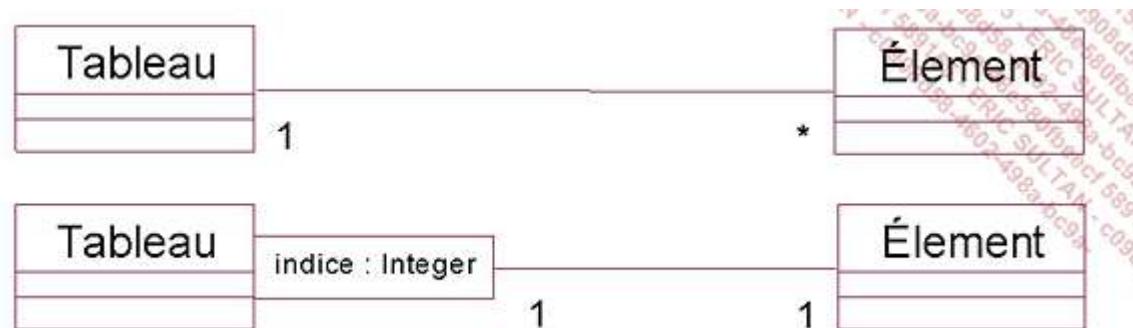
- La qualification d'une instance est une valeur ou un ensemble de valeurs qui permettent de retrouver cette instance
- Une qualification est souvent un index, par exemple pour retrouver un élément dans un tableau, ou une clef, par exemple pour retrouver une ligne dans une base de données relationnelle
- Cette qualification est alors insérée à l'autre extrémité sous la forme d'un ou de plusieurs attributs où les instances de la classe 2 sont qualifiées au niveau de la classe 1



# Les associations entre objets

## □ Exemple :

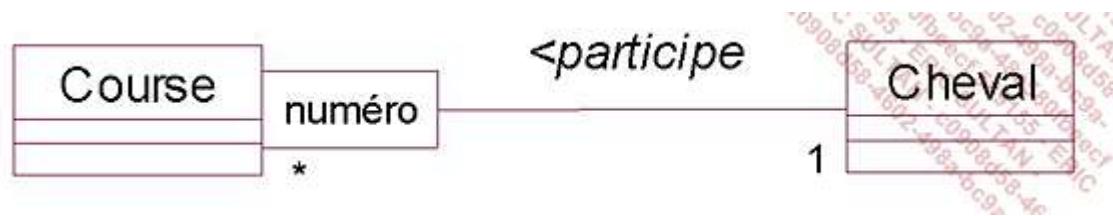
- *Un tableau est constitué d'éléments*
- *La première ne fait pas appel à la qualification tandis que, dans la seconde, le qualificateur indice permet de retrouver un seul élément du tableau ; par conséquent, la cardinalité maximale passe à un.*



# Les associations entre objets

## □ Exemple :

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



# Les associations entre objets

## □ L'expression de contraintes sur les associations :

- UML offre l'expression des contraintes, grâce à certaines constructions de la modélisation *objet* que nous avons déjà étudiées : les cardinalités, le type d'un attribut, etc.
- Pour d'autres contraintes, UML propose de les exprimer en langage naturel
- Il existe également le langage OCL, langage de contraintes *objet* sous forme de conditions logiques
- Ce langage OCL fait partie de l'ensemble de la notation UML
- Qu'elles soient écrites en langage naturel ou en OCL, ces deux types de contraintes sont représentés dans des notes incluses à l'intérieur du diagramme de classes
- Les contraintes écrites en OCL s'expriment sur la valeur des attributs et des rôles (extrémités des associations).
- Une contrainte doit avoir une valeur logique (vrai ou faux).

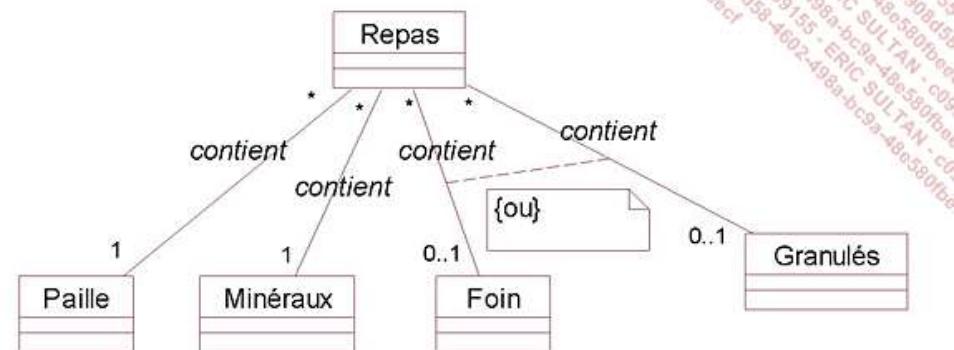
# Les associations entre objets

□ La syntaxe d'une expression de chemin est la suivante :

- Classe.role.role. .... .role.attribut
- Classe.role.role. .... .role.méthode

□ Exemple :

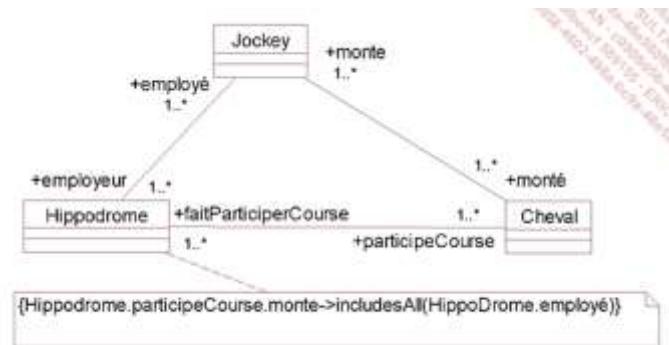
- *Un repas pour un cheval contient les éléments suivants :*
  - paille
  - minéraux
  - foin ou granulés



# Les associations entre objets

## □ Exemple :

- *Un hippodrome fait courir certains chevaux*
- *Il doit alors recruter les jockeys choisis par les propriétaires de ces chevaux pour les monter*
- *Ceci peut être reformulé ainsi : L'ensemble de jockeys recrutés doit être inclus dans l'ensemble de tous les jockeys qui montent les chevaux participant aux courses de l'hippodrome.*
- *En OCL, cette contrainte s'écrit ainsi :*
  - Hippodrome.participeCourse.monte ->includesAll(Hippodrome.employé)



# Les associations entre objets

## □ Les objets composés :

- Il existe deux formes de compositions, forte ou faible, que nous allons examiner maintenant
- La composition forte ou composition :
  - La composition forte est la forme de composition telle que les composants sont une partie de l'objet composé
  - Chaque composant ne peut ainsi être partagé entre plusieurs objets composés.
  - La cardinalité maximale, au niveau de l'objet composé, est donc obligatoirement de un
  - La suppression de l'objet composé entraîne la suppression de ces composants



# Les associations entre objets

## □ Exemple :

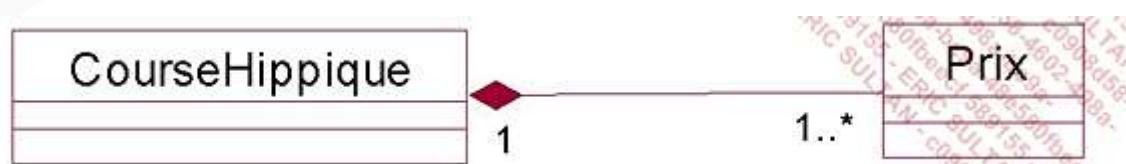
- *Un cheval est composé d'un cerveau.*
- *Le cerveau n'est pas partagé.*
- *La mort du cheval entraîne la mort de son cerveau. Il s'agit donc d'une association de composition.*



# Les associations entre objets

## □ 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*



# Les associations entre objets

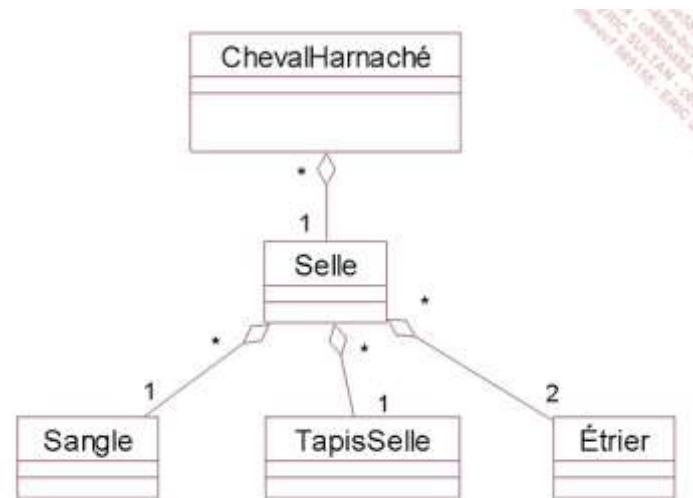
## □ La composition faible ou agrégation

- Dans le cas de l'agrégation, les composants peuvent être partagés par plusieurs composés (de la même association d'agrégation ou de plusieurs associations distinctes d'agrégation)
- La destruction du composé ne conduit pas à la destruction des composants.
- Lors d'une première phase de modélisation, il est possible d'utiliser seulement l'agrégation puis, plus tard, de déterminer quelles associations d'agrégation sont des associations de composition.

# Les associations entre objets

## □ Exemple :

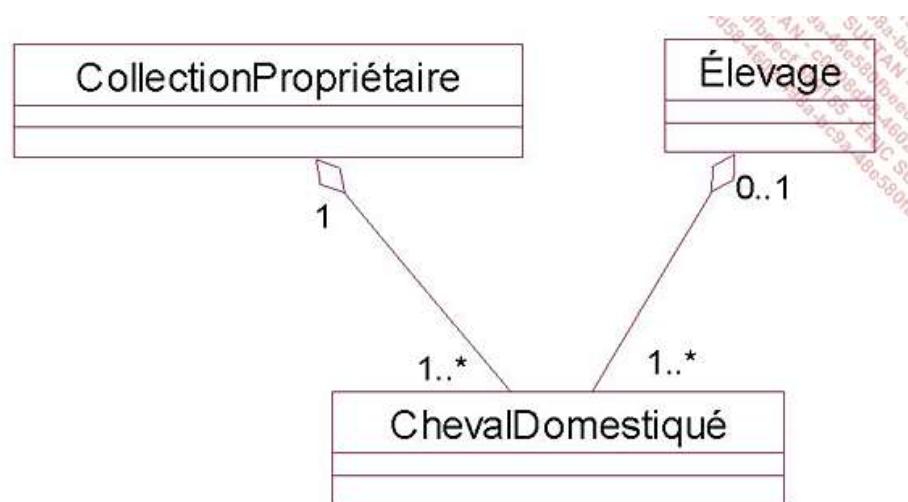
- *Un cheval harnaché est composé d'une selle*
- *Une selle est elle-même composée d'une sangle, d'étriers et d'un tapis de selle*
- *Cette composition relève de l'agrégation*
- *En effet, la perte du cheval n'entraîne pas la perte de ces objets et la perte de la selle n'entraîne pas la perte de ses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



# Les associations entre objets

## □ Exemple :

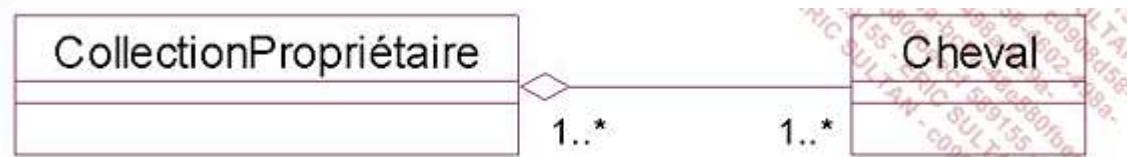
- *Un propriétaire équestre possède une collection de chevaux*
- *Un cheval domestiqué appartient à une seule collection et peut simultanément être confié ou non à un élevage*
- *Il peut être alors composant des deux agrégations*



# Les associations entre objets

## □ Exemple :

- *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*



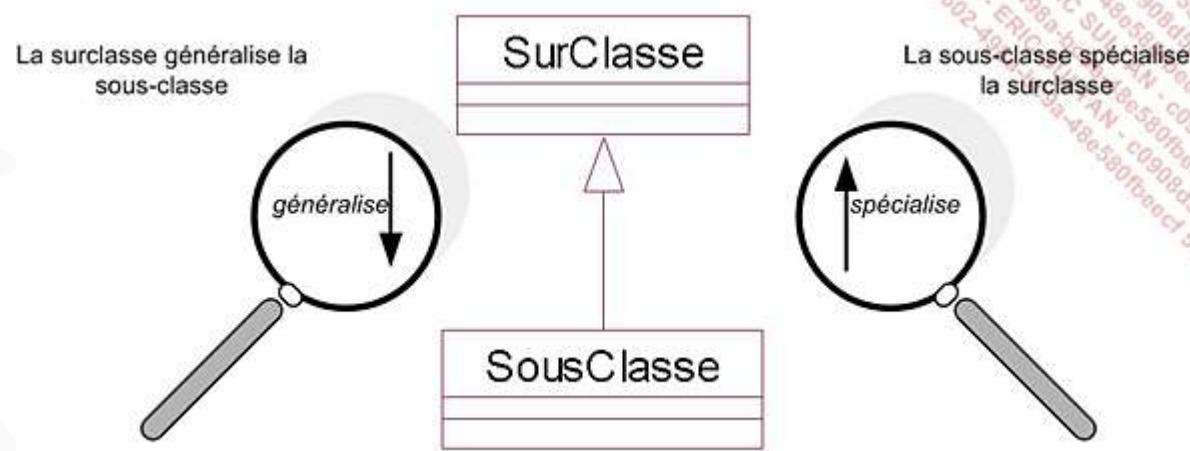
# Les associations entre objets

## □ Différences entre composition et agrégation

	Agrégation	Composition
Représentation	Losange transparent	Losange noir
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

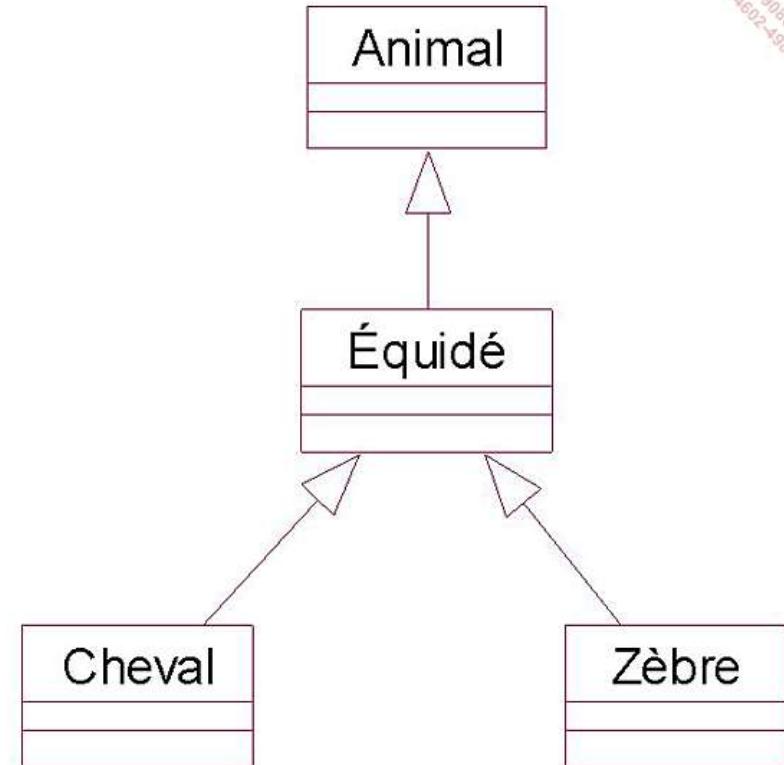
# La relation de généralisation / spécialisation entre les classes

- Classes plus spécifiques et classes plus générales :
  - Une classe est plus spécifique qu'une autre si toutes ses instances sont également instances de cette autre classe
  - La classe plus spécifique est dite sous-classe de l'autre classe
  - Cette dernière, plus générale, est dite surclasse



# La relation de généralisation / spécialisation entre les classes

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



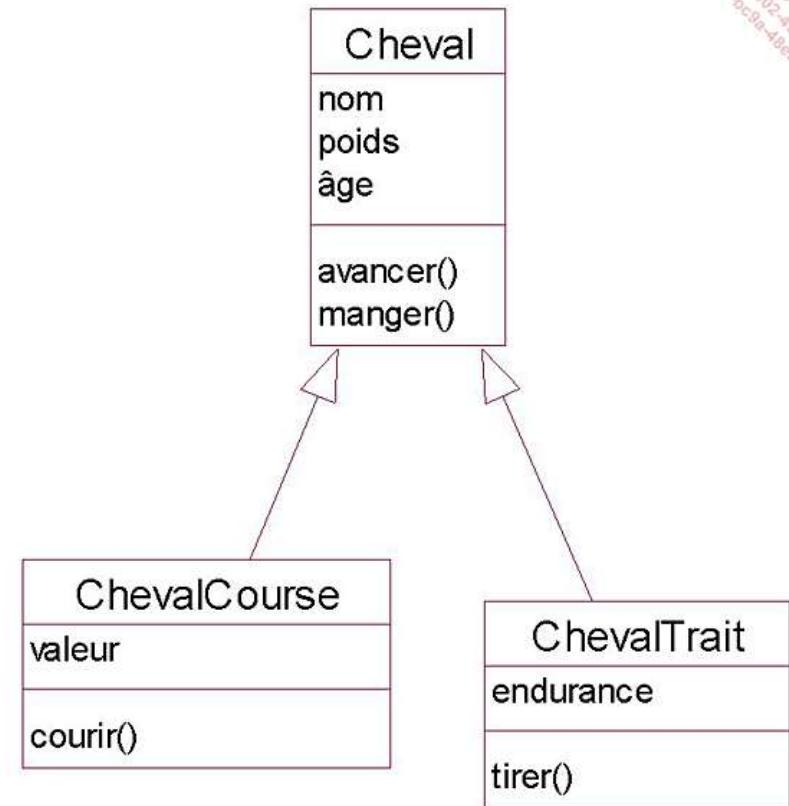
# La relation de généralisation / spécialisation entre les classes

## □ L'héritage :

- Les instances d'une classe sont aussi instances de sa ou de ses surclasses
- En conséquence, elles profitent des attributs et des méthodes définis dans la ou les surclasses, en plus des attributs et des méthodes introduits au niveau de leur classe
- Cette faculté s'appelle l'héritage, c'est-à-dire qu'une classe hérite des attributs et méthodes de ses surclasses pour en faire bénéficier ses instances

# La relation de généralisation / spécialisation entre les classes

- Exemple :
  - *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*



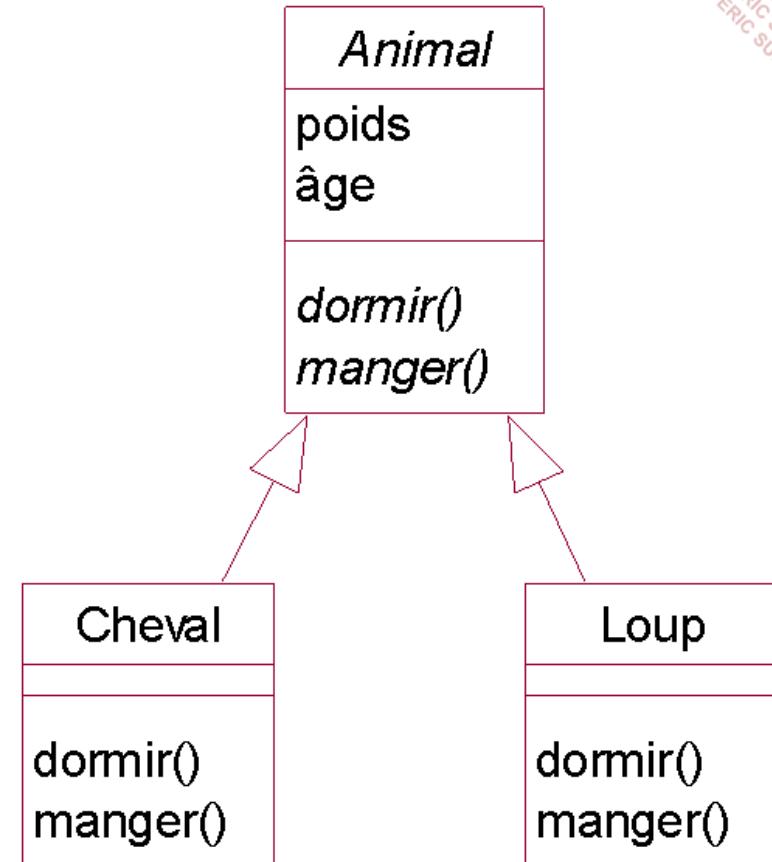
# La relation de généralisation / spécialisation entre les classes

## □ Classes concrètes et abstraites :

- Une classe concrète possède des instances.
- Elle constitue un modèle complet d'objet (tous les attributs et méthodes sont complètement décrits)
- À l'opposé, une classe abstraite ne peut pas posséder d'instance directe car elle ne fournit pas une description complète.
- Elle a pour vocation de posséder des sous-classes concrètes et sert à factoriser des attributs et méthodes communs à ses sous-classes.
- Souvent, la factorisation de méthodes communes aux sous-classes se traduit par la seule factorisation de la signature.
- Une méthode introduite dans une classe avec sa seule signature et sans code est appelée une méthode abstraite.
- En UML, une classe ou une méthode abstraite sont représentées par le stéréotype «*abstract*».
- Graphiquement, celui-ci est représenté soit explicitement, soit implicitement avec une mise en italiques du nom de la classe ou de la méthode.

# La relation de généralisation / spécialisation entre les classes

- Exemple :
  - *Un animal peut dormir ou manger mais de façon distincte selon la nature de l'animal.*
  - *Ces méthodes possèdent pour unique description, au niveau de la classe Animal, leur signature. Il s'agit donc de méthodes abstraites.*

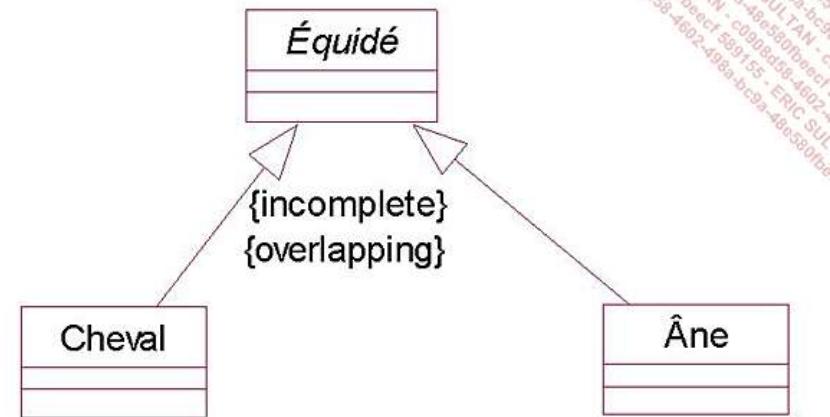


# La relation de généralisation / spécialisation entre les classes

- Expression de contraintes sur la relation d'héritage :
  - 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

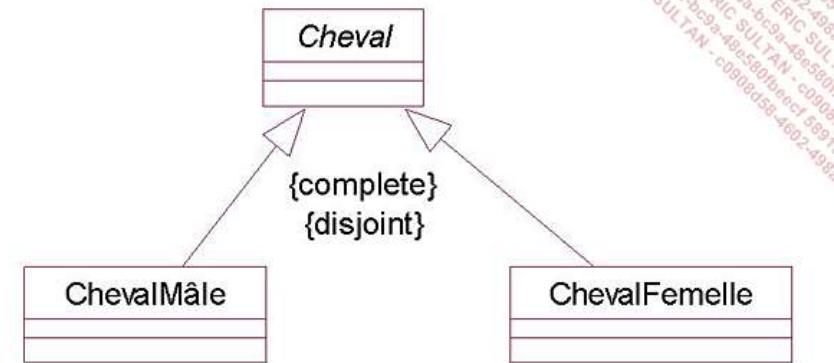
# La relation de généralisation / spécialisation entre les classes

- Exemple :
  - 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.
  - De ce fait, la figure fait ressortir les contraintes {incomplete} et {overlapping}.



# La relation de généralisation / spécialisation entre les classes

- Exemple :
  - Ces deux sous-classes couvrent la classe des chevaux et il n'existe aucun cheval qui soit à la fois mâle et femelle.
  - De ce fait, la figure fait ressortir les contraintes {complete} et {disjoint}.



# La relation de généralisation / spécialisation entre les classes

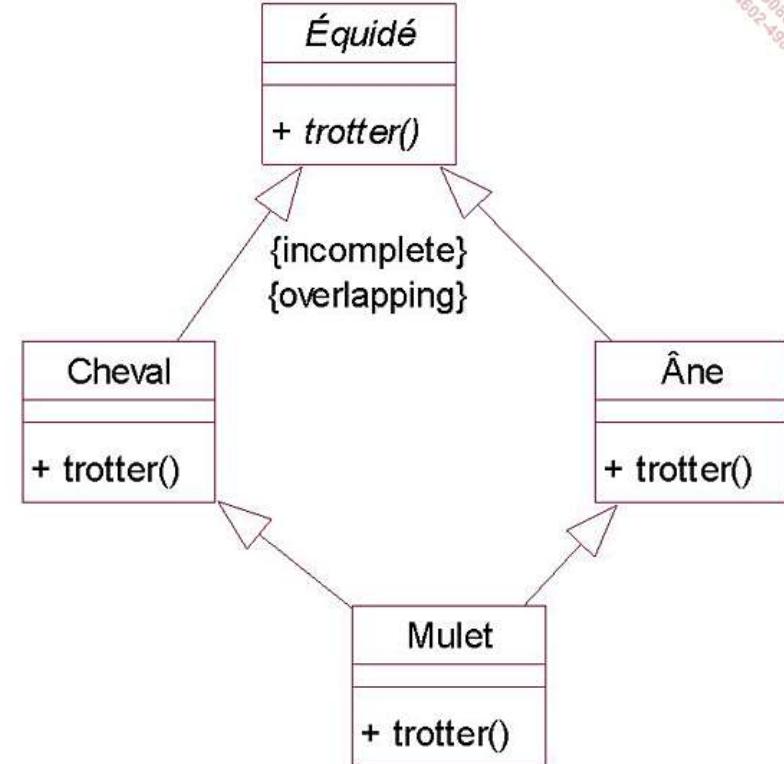
## □ L'héritage multiple :

- L'héritage multiple en UML consiste à ce qu'une sous-classe hérite de plusieurs supclasses
- Il pose un seul problème : il engendre un conflit lorsqu'une même méthode, c'est-à-dire de même signature, est héritée plusieurs fois dans la sous-classe
- En effet, lors de la réception d'un message appelant cette méthode, il faut définir un critère pour en choisir une parmi toutes celles qui sont héritées
- Une solution, dans ce cas, consiste à redéfinir la méthode dans la sous-classe afin de supprimer le conflit

# La relation de généralisation / spécialisation entre les classes

- Exemple :

- Illustrons la méthode *trotter*, abstraite dans la classe *Équidé*, rendue concrète dans ses sous-classes immédiates, et redéfinie dans la sous-classe fruit de leur héritage multiple.
- En effet, lorsque son cavalier lui demande de trotter, un mulet répond à la fois par des réactions de cheval et des réactions d'âne.
- Il peut être dangereux comme un cheval et agaçant comme un âne.



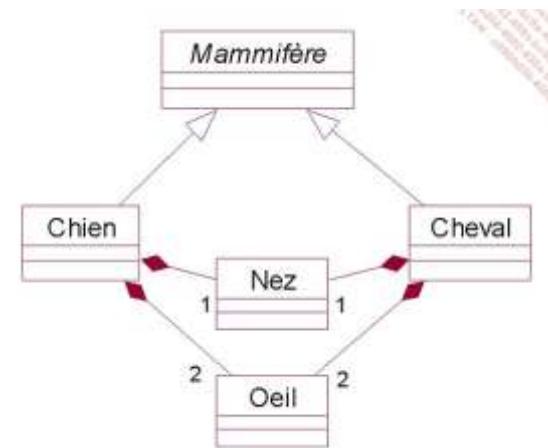
# La relation de généralisation / spécialisation entre les classes

## □ Factorisation des relations entre objets :

- Nous avons vu qu'une classe abstraite sert à factoriser les attributs et méthodes de plusieurs sous-classes
- Il est également possible, parfois, de factoriser l'extrémité d'une association dans une surclasse pour rendre le diagramme plus simple

## □ Exemple :

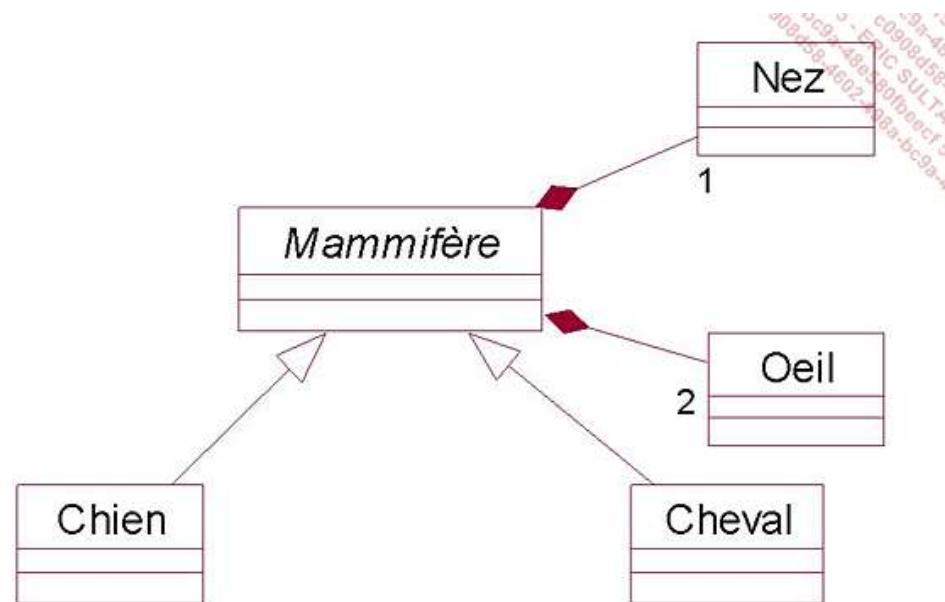
- *Un cheval comme un loup possède deux yeux et un nez*



# La relation de généralisation / spécialisation entre les classes

## □ Exemple (suite) :

- *La classe abstraite Mammifère surclasse des classes Chien et Loup, les deux associations de composition sont factorisées*



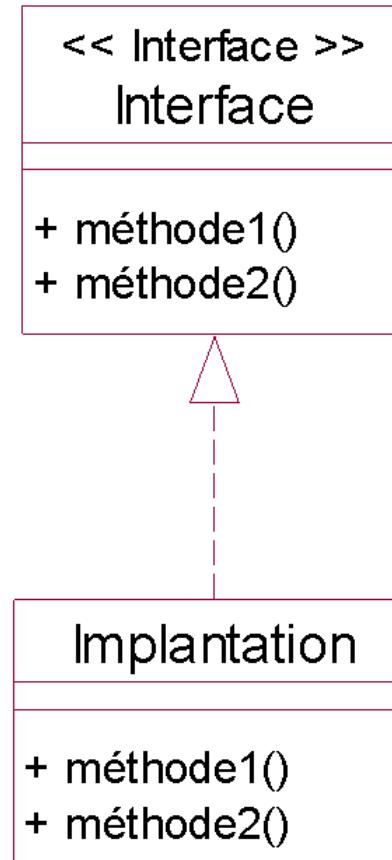
# La relation de généralisation / spécialisation entre les classes

## □ Interface :

- Une interface est une classe totalement abstraite, c'est-à-dire sans attribut et dont toutes les méthodes sont abstraites et publiques
- Une telle classe ne contient aucun élément d'implantation des méthodes
- Graphiquement, elle est représentée comme une classe avec le stéréotype «interface»

# La relation de généralisation / spécialisation entre les classes

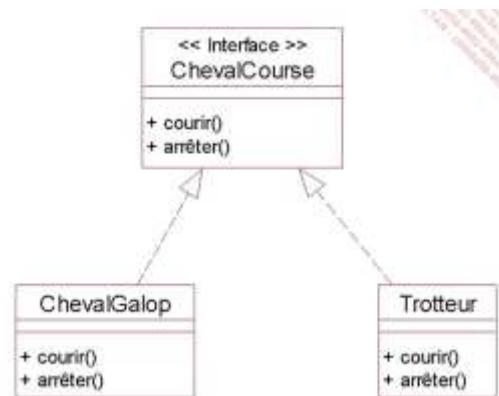
- L'implantation des méthodes est réalisée par une ou plusieurs classes concrètes, sous-classes de l'interface
- Dans ce cas, la relation d'héritage qui existe entre l'interface et une sous-classe d'implantation est appelée *relation de réalisation*
- Graphiquement, elle est représentée par un trait pointillé au lieu d'un trait plein pour une relation d'héritage entre deux classes



# La relation de généralisation / spécialisation entre les classes

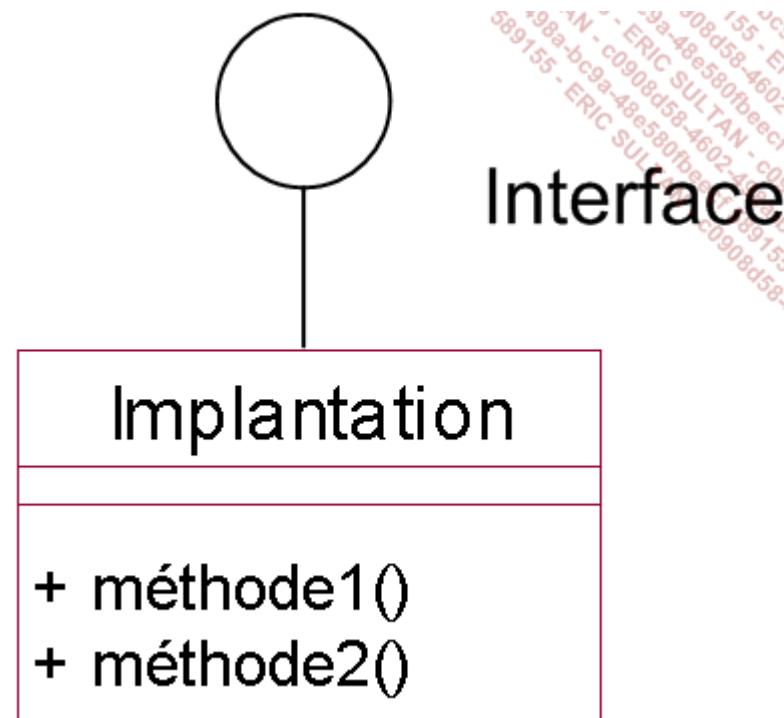
## □ Exemple :

- *Un cheval de course peut être considéré comme une interface. Celle-ci est composée de plusieurs méthodes : courir, arrêter, etc.*
- *L'implantation peut ensuite différer. Une course de galop ou de trot se fait seulement avec des chevaux entraînés pour l'une ou l'autre de ces courses.*
- *Pour des chevaux de galop, courir signifie galoper.*
- *Pour des chevaux de trot (un trotteur), courir signifie trotter.*
- *Tous deux répondent à l'interface ChevalCourse mais avec une implantation différente due à leur entraînement.*



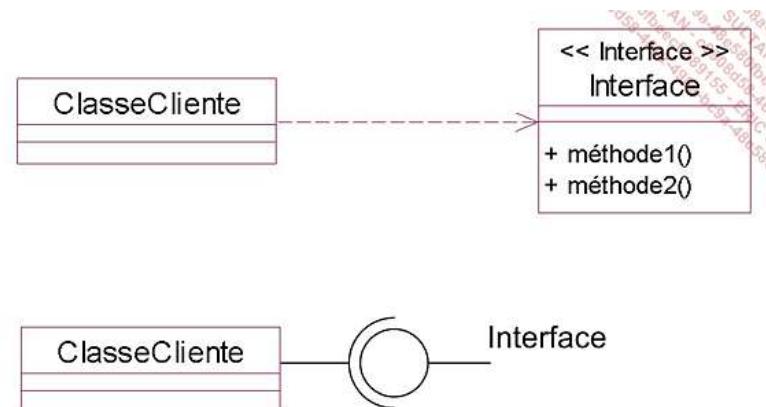
# La relation de généralisation / spécialisation entre les classes

- Il est également possible de représenter la relation de réalisation par une lollipop (une *lollipop* est une sucette)



# La relation de généralisation / spécialisation entre les classes

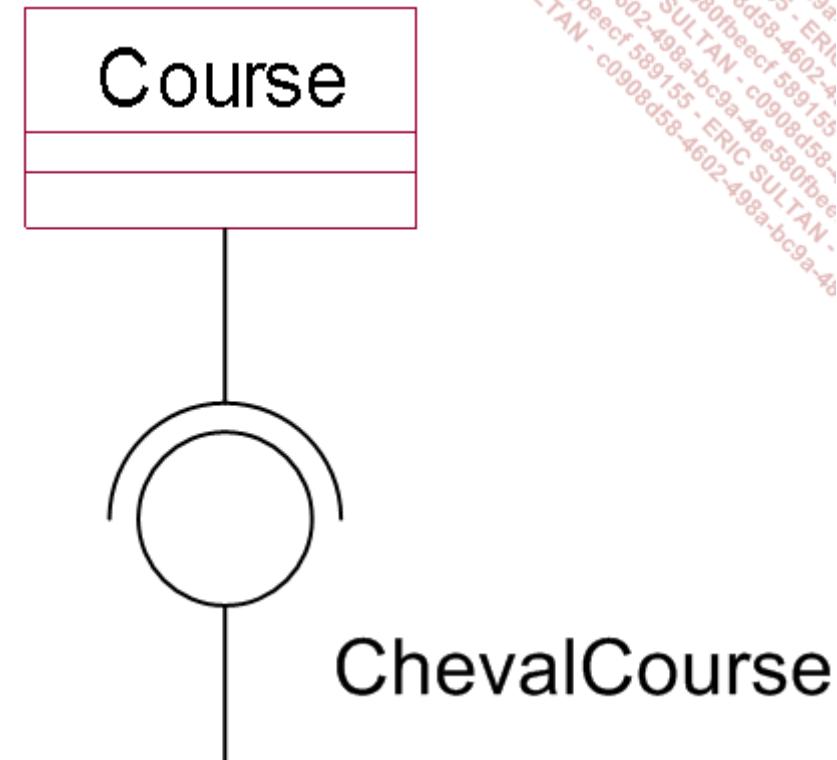
- Une classe peut également dépendre d'une interface pour réaliser ses opérations.
- Cette dernière est alors employée comme type au sein de la classe (attribut, paramètre de l'une des méthodes ou variable locale de l'une des méthodes)
- Une classe qui dépend d'une interface en est sa cliente



# La relation de généralisation / spécialisation entre les classes

- Exemple :

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

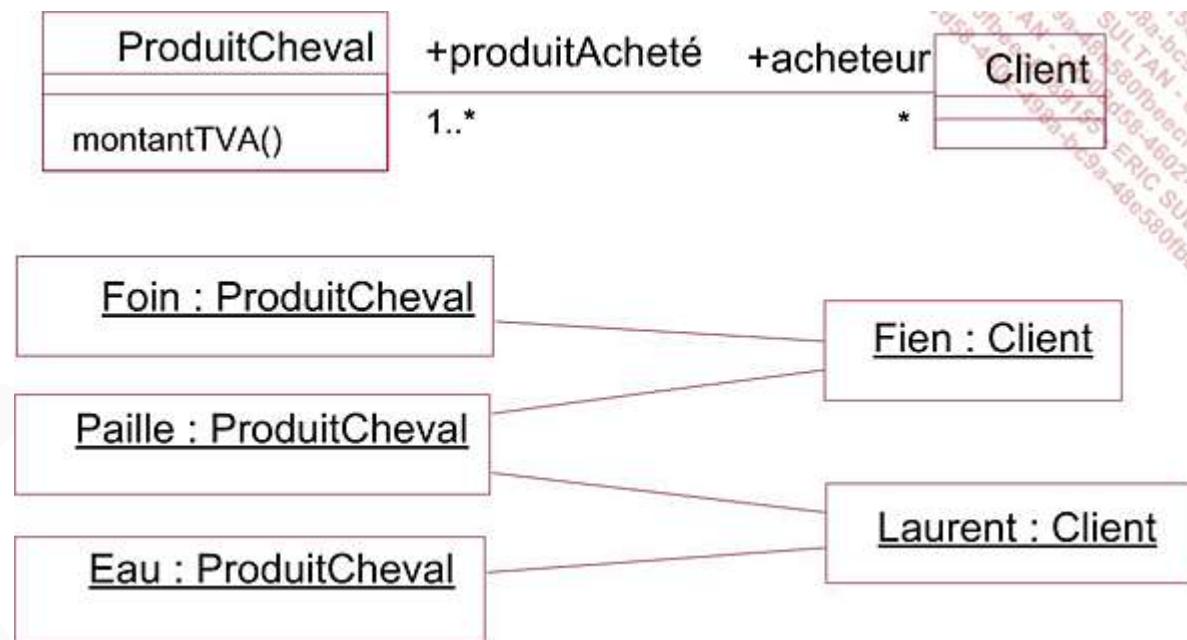


# Le diagramme des objets ou instances

- Le diagramme des classes est une représentation statique du système
- Le diagramme des objets montre, 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 d'une instance est de la forme : *nomInstance : nomClasse*
- Le nom de l'instance est optionnel
- La valeur d'un attribut est de la forme : *nomAttribut = valeurAttribut*
- Enfin, les liens entre instances sont représentés par de simples traits continus

# Le diagramme des objets ou instances

□ Exemple :



# Conclusion

- Les classes décrivent les attributs et les méthodes de leurs instances, ainsi que les attributs et méthodes de classe
- Les associations entre objets sont obligatoires lors de la conception d'un diagramme de classes
- Elles constituent le socle de l'interaction des instances par les envois de message lorsque le système est actif
- Les relations d'héritage entre les classes sont également indispensables
- Elles favorisent la factorisation d'éléments communs et permettent ainsi de réduire conséquemment la taille du diagramme
- Enfin, l'expression des contraintes en langage naturel ou en OCL conduit à enrichir encore la sémantique exprimée dans le diagramme

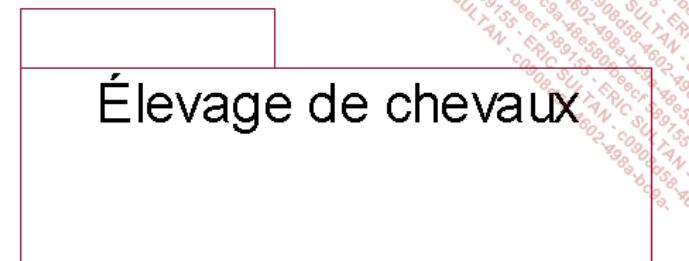
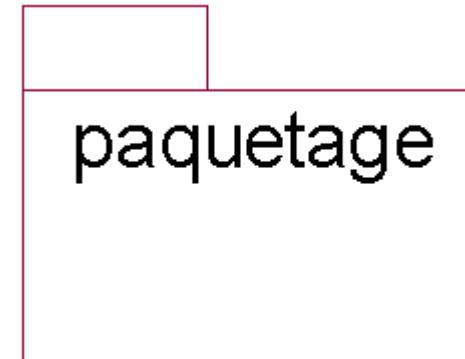
# LA STRUCTURATION DES ÉLÉMENTS DE MODÉLISATION

# Introduction

- UML 2 décrit les paquetages à l'aide d'un diagramme spécifique
- Un paquetage est un regroupement d'éléments de modélisation : classes, composants, cas d'utilisation, autres paquetages, etc...
- Les paquetages d'UML sont utiles lors de la modélisation de systèmes importants pour en regrouper les différents éléments
- Ce regroupement structure ainsi la modélisation
- *Remarque : En UML 1, les paquetages faisaient partie du diagramme de classes et regroupaient exclusivement des ensembles de classes*

# Paquetage et diagramme de paquetage

- Un paquetage est représenté par un dossier
- Il constitue un ensemble d'éléments de modélisation UML
- Exemple :
  - Illustrons le paquetage regroupant les différents éléments de modélisation d'un élevage de chevaux



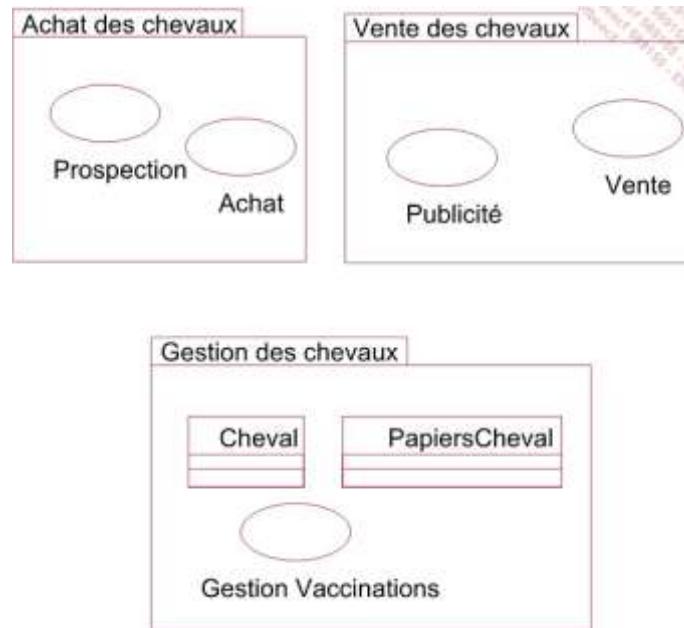
# Paquetage et diagramme de paquetage

- Le contenu d'un paquetage est décrit par un diagramme de paquetage
- Celui-ci représente les différents éléments du paquetage avec leur propre représentation graphique
- Ceux-ci peuvent être des classes, des composants, des cas d'utilisation, d'autres paquetages, etc...
- Il est possible d'inclure directement les éléments d'un paquetage à l'intérieur du dossier qui le représente

# Paquetage et diagramme de paquetage

## □ 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*



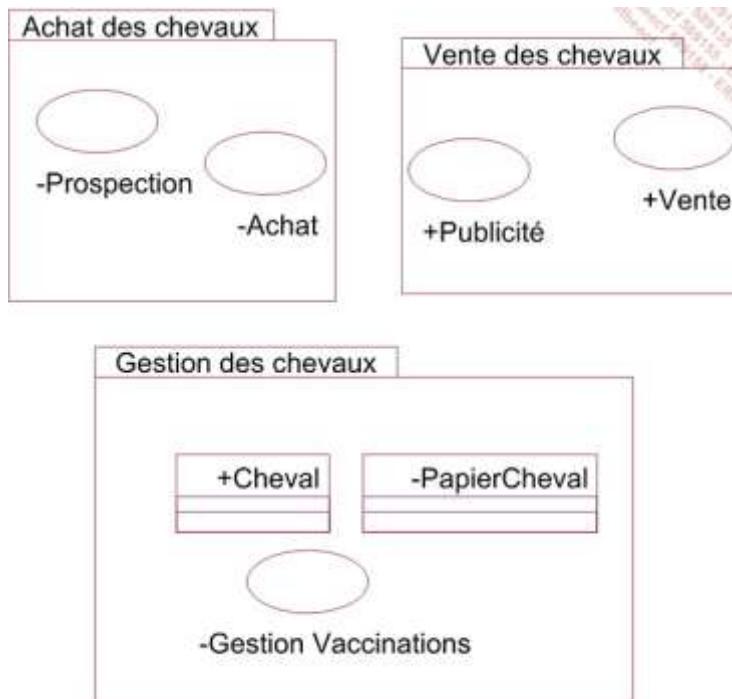
# Paquetage et diagramme de paquetage

- Chaque élément inclus dans un paquetage peut être accessible à l'extérieur ou encapsulé à l'intérieur de celui-ci
- Par défaut, un élément est accessible à l'extérieur
- L'encapsulation est représentée par un signe plus ou un signe moins, précédant le nom de l'élément
- Le signe plus signifie qu'il n'y pas d'encapsulation et que l'élément est visible en dehors du paquetage
- Le signe moins signifie que l'élément est encapsulé et n'est pas visible à l'extérieur

# Paquetage et diagramme de paquetage

## □ Exemple :

- *Illustrons le paquetage "Élevage de chevaux" pour lequel l'encapsulation a été précisée*



# Les associations entre les paquetages

- Un élément de la modélisation ne peut être présent que dans un seul paquetage
- Pour qu'un paquetage puisse exploiter les éléments d'un autre paquetage, il existe deux types d'associations :
  - L'association 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
  - L'association 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.

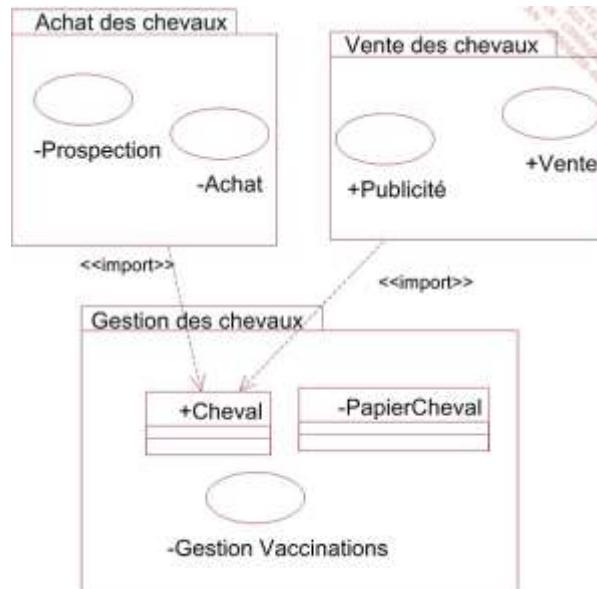
# Les associations entre les paquetages

- Il n'est possible d'importer ou d'accéder à un élément que si celui-ci est spécifié visible dans le paquetage d'origine
- Ces deux associations peuvent également s'appliquer à un paquetage complet : elles importent ou accèdent à l'intégralité des éléments du paquetage d'origine qui sont définis visibles
- Ces deux associations sont des associations de dépendance qui sont spécialisées à l'aide du stéréotype «import» ou «access»

# Les associations entre les paquetages

## □ 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*



# Conclusion

- Les paquetages sont utiles pour structurer la modélisation d'un système 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
- Cette structuration de la modélisation présente l'avantage de faire partie de la notation UML
- Par conséquent, elle est standardisée

# **LA MODÉLISATION DU CYCLE DE VIE DES OBJETS**

# Introduction

- Le cycle de vie d'un objet représente les différentes étapes ou états que celui-ci va suivre pour concourir, au sein du système, à la réalisation d'un objectif
- Un état correspond à un moment d'activité ou d'inactivité de l'objet
- L'inactivité se produit lorsque l'objet attend que d'autres objets finissent une activité
- Nous étudierons la notion d'événement, signal qui fait changer l'objet d'état
- Le changement d'état consiste à franchir une transition

# Introduction

- Le diagramme d'états-transitions illustre l'ensemble des états du cycle de vie d'un objet séparés par des transitions
- Chaque transition est associée à un événement
- Nous étudierons en détail la notion de signal ainsi que la possibilité d'en envoyer à différents moments
- Nous verrons qu'il est possible d'associer des conditions au franchissement des transitions et de gérer ainsi des alternatives
- La notion d'état composé sera examinée afin de simplifier l'écriture du diagramme d'états-transitions
- Nous verrons qu'il est possible de disposer de sous-états évoluant en parallèle
- Enfin, nous présenterons le diagramme de *timing* qui décrit les changements d'état d'un objet lorsqu'ils sont exclusivement fonction du temps

# La notion d'état

- L'état d'un objet correspond à un moment de son cycle de vie
- Pendant qu'il se trouve dans un état, un objet peut se contenter d'attendre un signal provenant d'autres objets : il est alors inactif
- Il peut également être actif et réaliser une activité
- Une activité est l'exécution d'une série de méthodes et d'interactions avec d'autres objets.
- 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*

# La notion d'état

- L'ensemble des états du cycle de vie d'un objet contient un état initial
- Celui-ci correspond à l'état de l'objet juste après sa création
- Il peut également contenir un ou plusieurs états finaux
- Ceux-ci correspondent à une phase de destruction de l'objet
- Il arrive également qu'il n'y ait pas d'état final car un objet peut ne jamais être détruit

# Le changement d'état

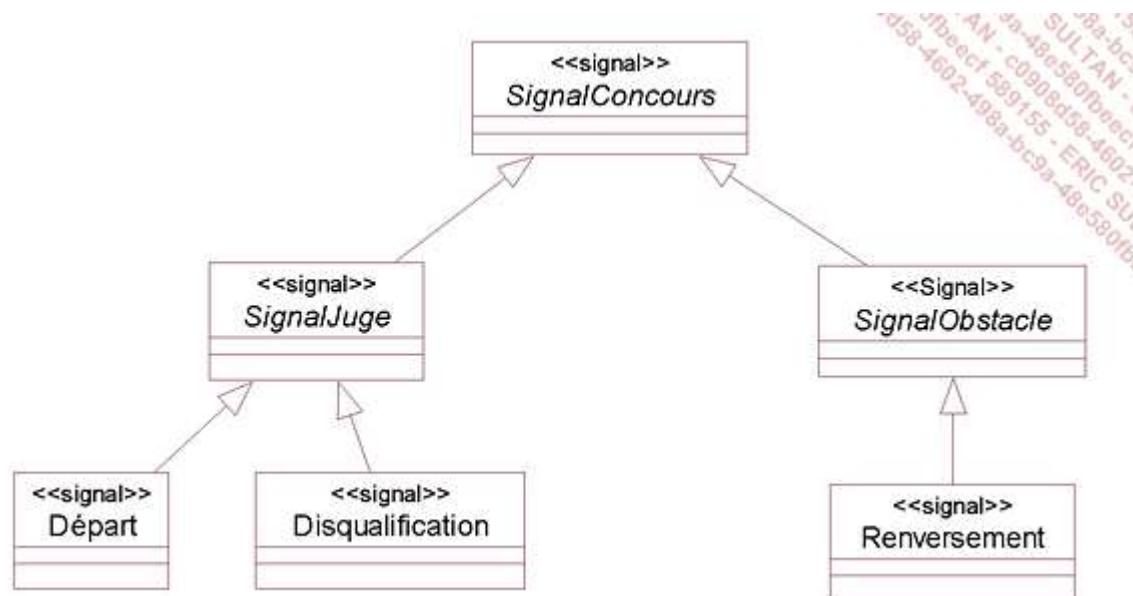
## □ La notion d'événement et de signal :

- Un événement est un fait qui déclenche le changement d'état
- Il est lié à la réception d'un signal par l'objet
- La réception d'un signal équivaut à la réception d'un message
- UML propose de décrire les signaux par des classes :
  - Chaque signal émis et reçu est alors une instance d'une classe de signaux
  - Pour les distinguer des autres classes, les classes de signaux sont décrites avec le stéréotype «signal»
  - Les attributs des objets d'une telle classe sont les paramètres de message
  - Cette description par des classes conduit à l'organisation des signaux en hiérarchies

# Le changement d'état

## □ Exemple :

- Illustrons la hiérarchie des signaux que peuvent recevoir le cavalier et son cheval. Il s'agit soit de signaux émis par le juge, soit par l'un des obstacles



# Le changement d'état

## □ La transition :

- Une transition est un lien orienté entre deux états qui exprime le fait que l'objet a la possibilité de passer de l'état d'origine de la transition à son état de destination
- Lorsque l'objet réalise ce passage de l'état d'origine à l'état de destination, la transition est alors franchie
- Une transition est généralement associée à un événement
- Dans ce cas, la transition est franchie si l'objet se trouve dans l'état d'origine de la transition et s'il reçoit l'événement
- Ce franchissement a lieu que l'objet soit actif ou non
- S'il est inactif, le franchissement a lieu immédiatement
- S'il est actif, le franchissement a lieu dès que l'activité associée à l'état est terminée

# Le changement d'état

- 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*
- 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*

# Le changement d'état

- 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 précédent cet obstacle. À chaque fois, le nombre de tentatives est augmenté de un*

# L'élaboration du diagramme d'états-transitions

- Le diagramme d'états-transitions représente le cycle de vie des instances d'une classe
- Il décrit les états, les transitions qui les lient et les événements qui provoquent le franchissement des transitions
- Un tel diagramme n'est utile que pour les objets qui ont un cycle de vie.
- D'autres objets, purement porteurs d'information, ne changent pas d'état au cours de leur vie. Pour ces objets, il est inutile de concevoir un diagramme d'états-transitions

# L'élaboration du diagramme d'états-transitions

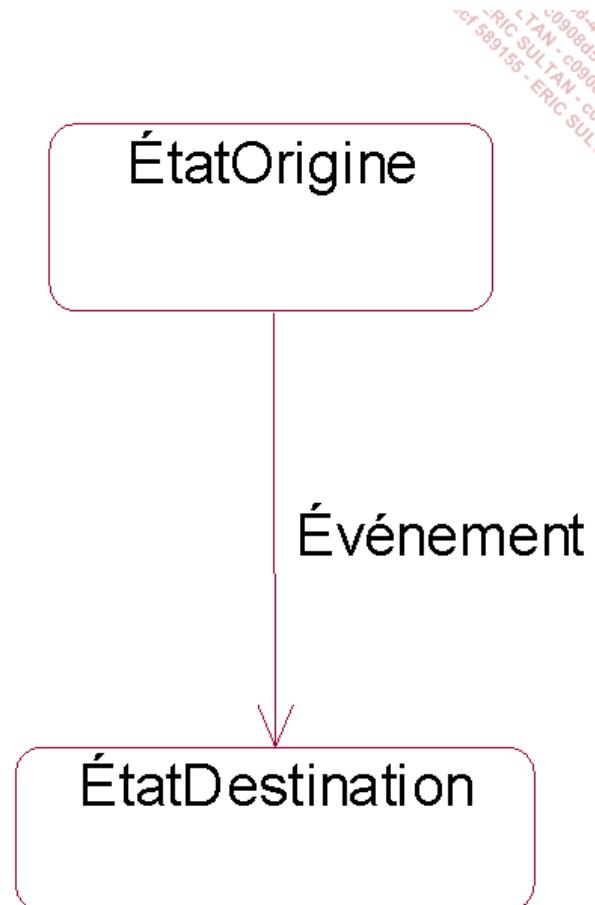
## □ La représentation graphique des éléments de base

- Un état est représenté par un rectangle aux coins arrondis contenant son nom
- Dans un diagramme d'états-transitions, le premier état correspond à l'état initial de l'objet à l'issue de sa phase de création. Cet état est unique dans un diagramme d'états-transitions : l'état initial est représenté par un point noir
- Un état final correspond à une étape où l'objet n'est plus nécessaire dans le système et où il est détruit. Tous les objets n'ont pas d'état final. C'est notamment le cas des objets permanents dans le système : un état final est représenté par un point noir entouré d'un cercle



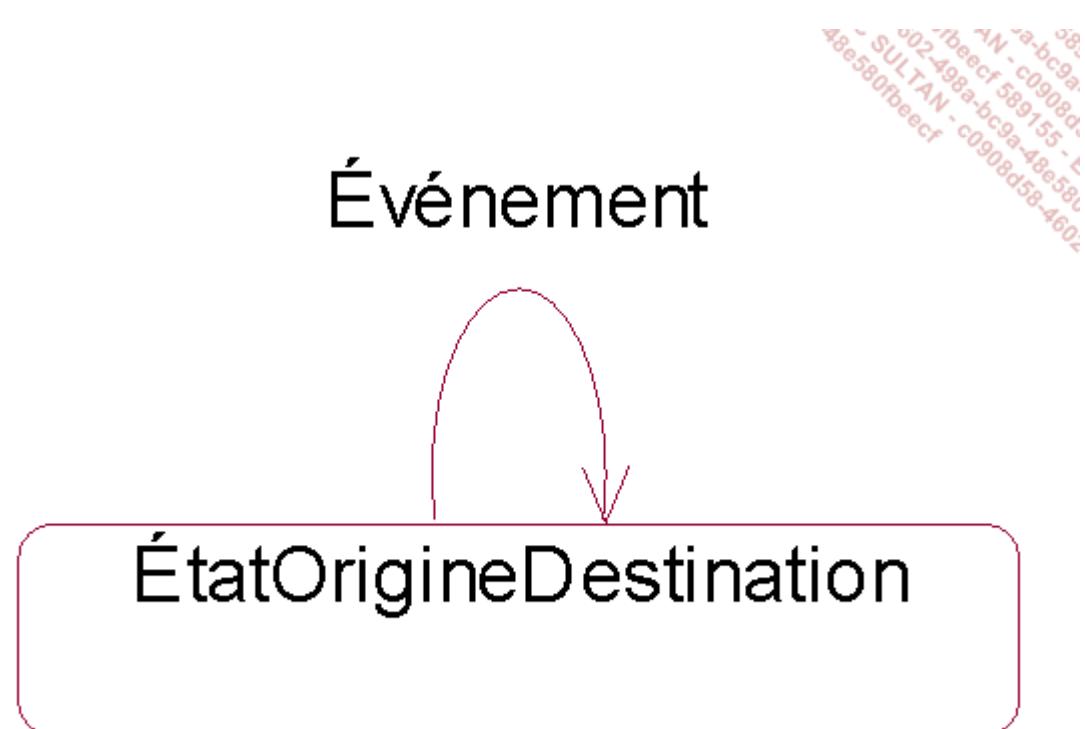
# L'élaboration du diagramme d'états-transitions

- Une transition entre deux états est représentée par un trait droit fléché reliant ces deux états
- L'événement qui détermine le franchissement de la transition est indiqué à proximité de la transition
- Si la transition est automatique, aucun événement n'est indiqué



# L'élaboration du diagramme d'états-transitions

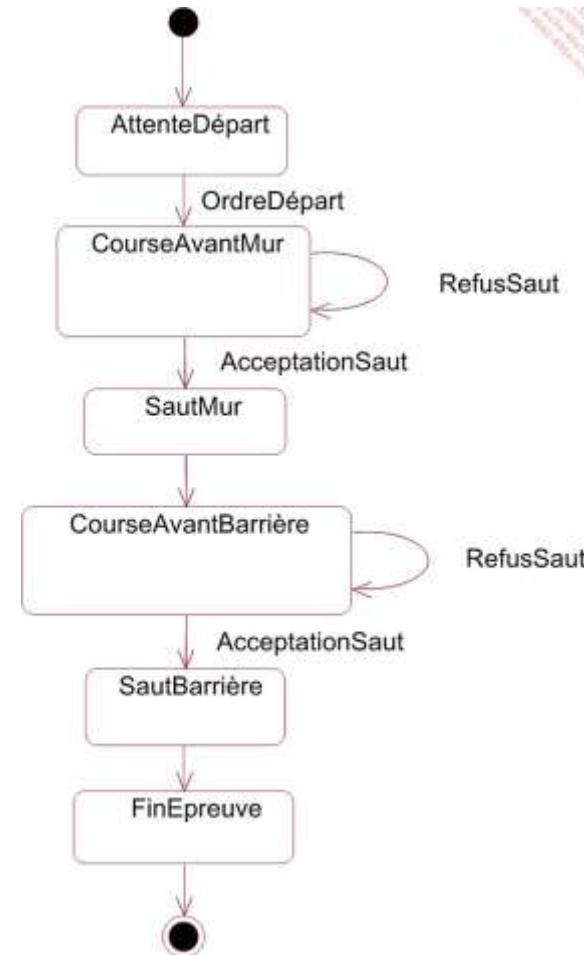
- Une transition réflexive possède le même état d'origine et de destination



# L'élaboration du diagramme d'états-transitions

- Exemple :

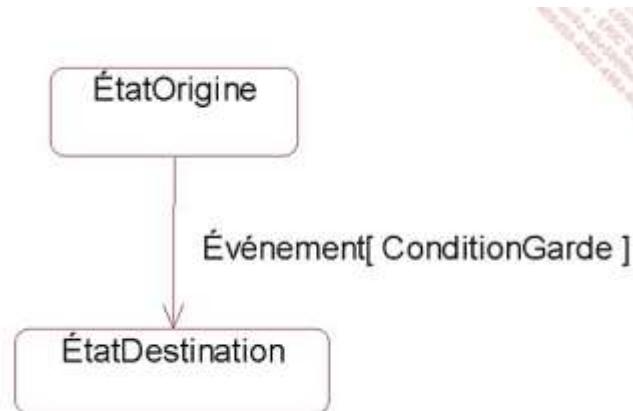
- *Dans un concours d'obstacles, l'épreuve consiste à demander à chaque concurrent de sauter deux ou trois obstacles différents*
- *Il arrive que le cheval refuse de sauter un obstacle. Le concurrent peut alors recommencer le saut*
- *Illustrons le diagramme d'états-transitions décrivant une telle épreuve pour l'objet "concurrent de l'épreuve".*



# L'élaboration du diagramme d'états-transitions

## □ Les conditions de garde :

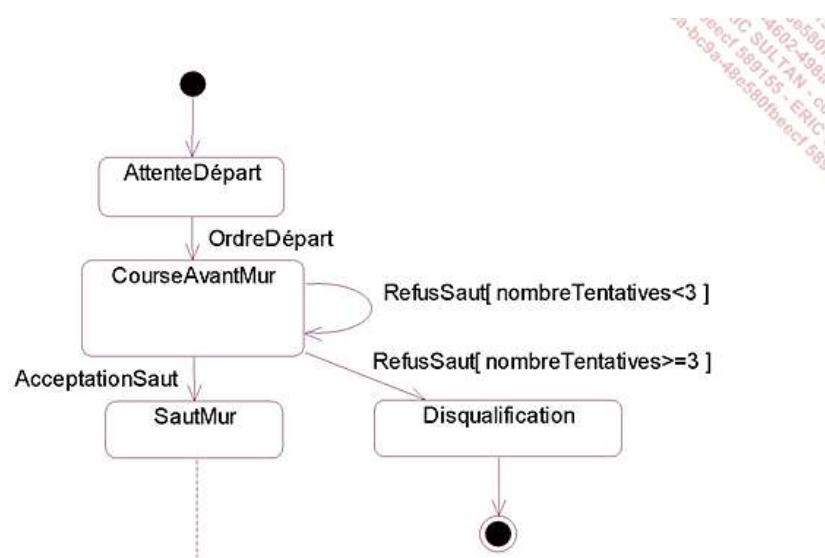
- Il est possible d'associer une condition à une transition, qui est alors appelée *condition de garde*
- Pour que la transition soit franchie, il faut que la condition soit remplie en plus de la réception de l'événement associé, si celui-ci existe.
- Une condition de garde est exprimée entre crochets. Si un événement est associé à la transition, la condition est exprimée à la droite du nom de l'événement



# L'élaboration du diagramme d'états-transitions

## □ Exemple :

- *En cas de refus de sauter un obstacle, le concurrent a le droit de recommencer deux fois. Il est donc disqualifié après la troisième tentative si elle se solde par un refus*
- *Illustrons la prise en compte de ce nombre maximal de refus*



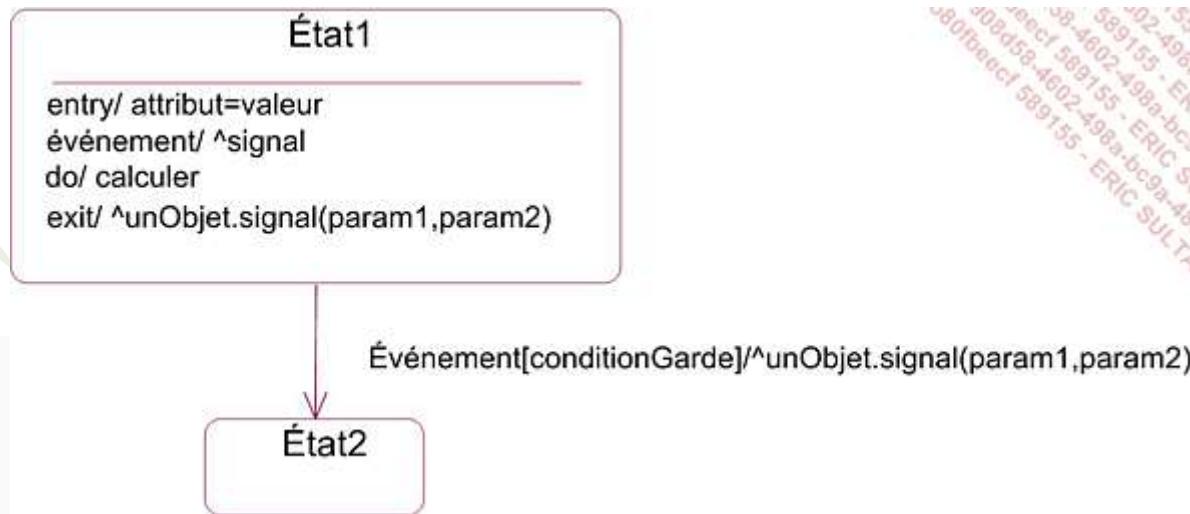
# L'élaboration du diagramme d'états-transitions

- Activités liées à un état ou à un franchissement de transition :
  - 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
  - Une activité est une série d'actions
  - Une action consiste à affecter une valeur à un attribut, créer ou détruire un objet, effectuer une opération, envoyer un signal à un autre objet ou à soi-même, etc...
  - On désignera l'autre objet par son nom comme dans les diagrammes d'interaction

# L'élaboration du diagramme d'états-transitions

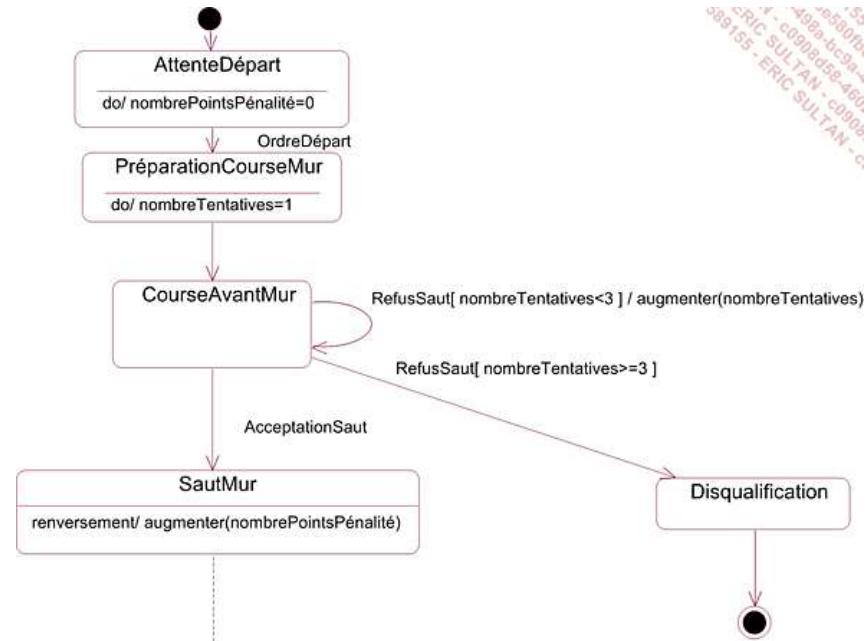
- Une activité précédée du mot clé *entry*/ est exécutée lors de l'entrée dans l'état
- Une activité précédée du nom d'un événement est exécutée si cet événement est reçu
- Le mot clé *do*/ introduit l'activité réalisée pendant l'état
- Une activité précédée du mot clé *exit*/ est exécutée lors de la sortie de l'état
- L'envoi d'un signal est précédé d'un ^ suivi du nom du signal
- Il est également possible de spécifier une activité lors du franchissement d'une transition, qu'il faut faire précéder d'un /, et à la suite de l'événement et de la condition de garde, s'ils existent

# L'élaboration du diagramme d'états-transitions



# L'élaboration du diagramme d'états-transitions

- Exemple :
  - Illustrons l'utilisation de ces 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 un, puis augmenté à chaque refus de sauter lors de la transition correspondante



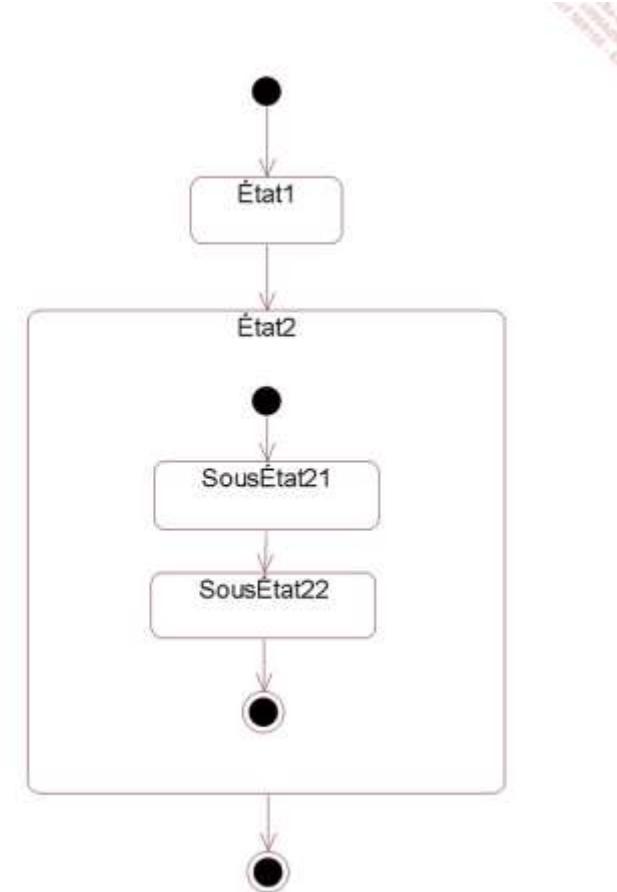
# L'élaboration du diagramme d'états-transitions

## □ Etats composés :

- Un état peut être décrit lui-même par un diagramme d'états-transitions
- Un tel état est appelé un état composé
- Les états qui le composent sont appelés sous-états.
- Le principe est simple : dès que l'objet passe dans l'état composé, il passe également dans le sous-état initial du diagramme interne d'états-transitions
- Si l'objet franchit une transition qui fait sortir de l'état composé, il quitte également les sous-états.

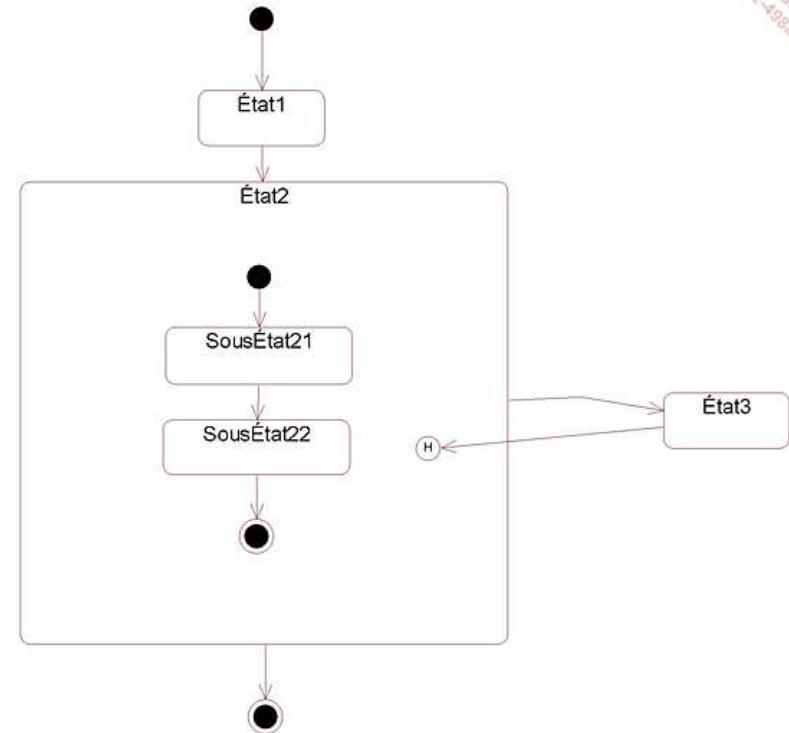
# L'élaboration du diagramme d'états-transitions

- Un diagramme interne d'états- transitions peut soit ne pas avoir d'état final, soit avoir un ou plusieurs états finaux



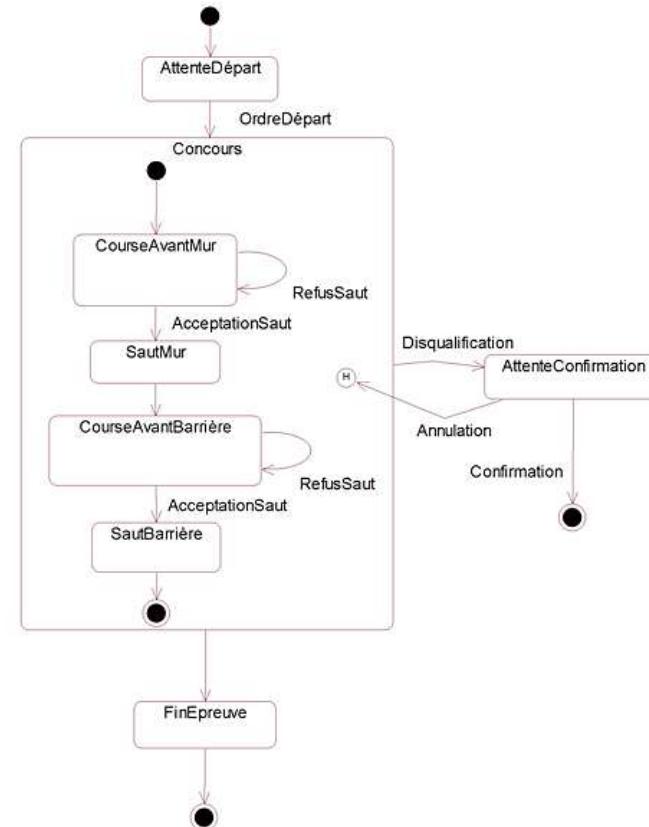
# L'élaboration du diagramme d'états-transitions

- Il est possible, lorsque un objet quitte un état composé, de mémoriser le sous-état actif pour y revenir
- Pour cela, il faut utiliser le sous-état spécial de mémoire H qui représente dans l'état composé le dernier sous-état actif mémorisé



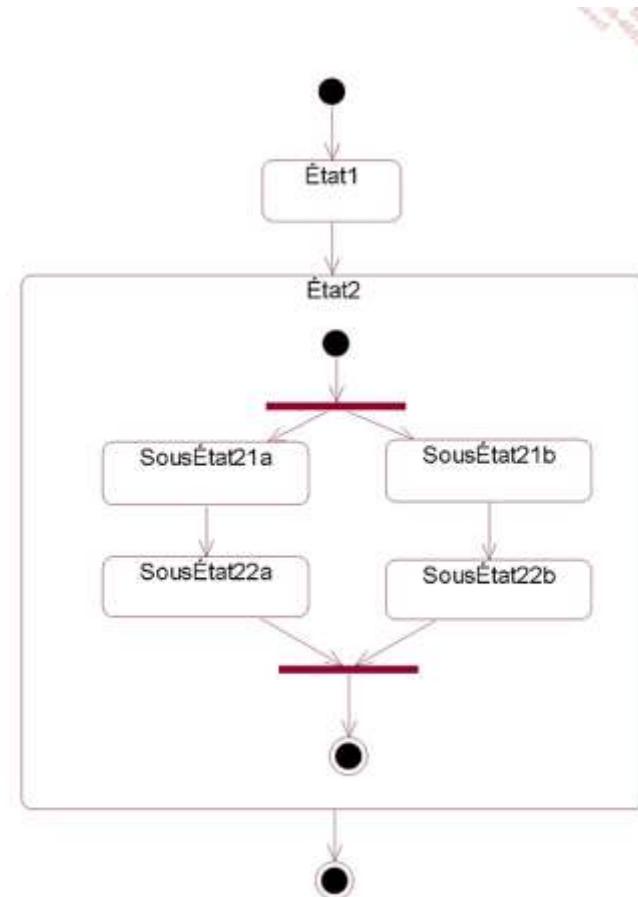
# L'élaboration du diagramme d'états-transitions

- Exemple :
  - Après l'ordre de départ et jusqu'au dernier saut, un concurrent est dans l'état *Concours*
  - À tout moment, il peut être disqualifié mais cette disqualification doit être confirmée (par exemple, en cas de contestation)
  - Si elle est annulée, l'épreuve repart de l'état dans lequel elle s'était arrêtée



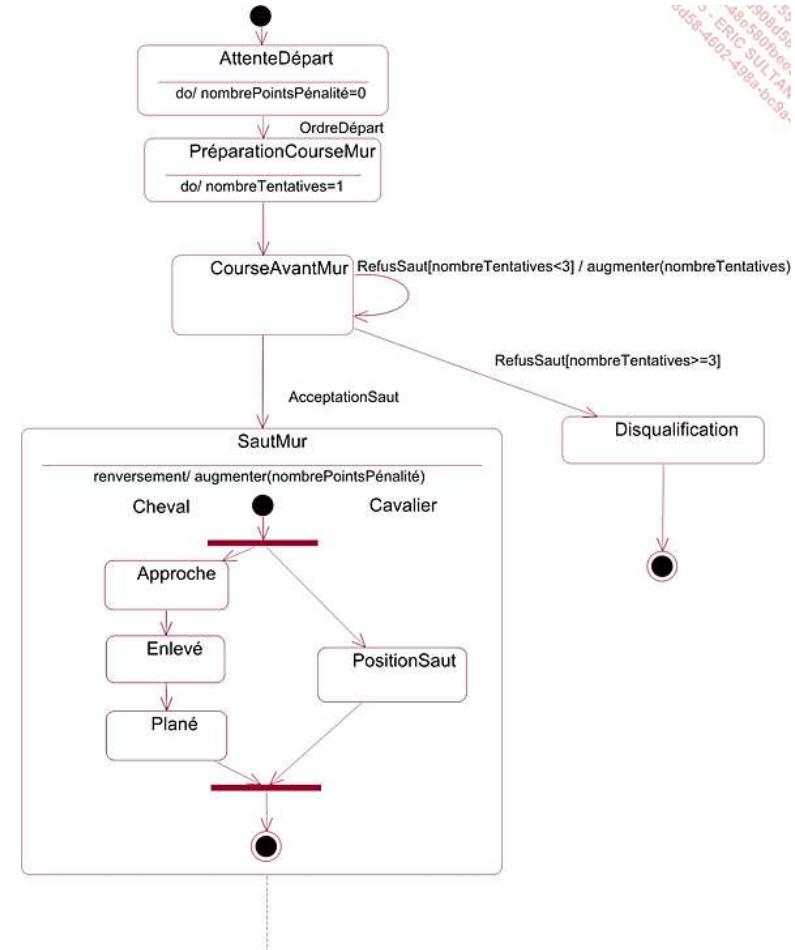
# L'élaboration du diagramme d'états-transitions

- Au sein d'un objet composé, il est possible d'avoir des sous-états qui évoluent en parallèle
- Pour cela, il existe une transition de type *fourche* qui possède plusieurs sous-états de destination
- Une fois franchie, l'objet se trouve dans tous les sous-états de destination
- La transition de type *synchronisation* possède plusieurs sous-états d'origine et un seul état de destination
- Il faut que l'objet se trouve dans tous les sous-états d'origine pour que la transition soit franchie



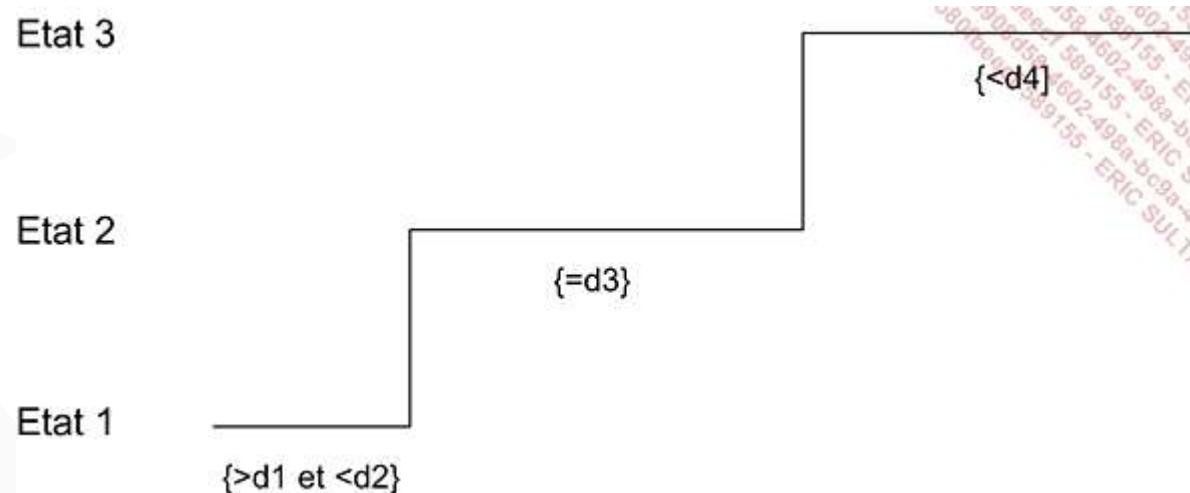
# L'élaboration du diagramme d'états-transitions

- Exemple :
  - Un saut peut être décomposé en sous-états qui sont différents pour le cheval et le cavalier mais qui ont lieu simultanément
  - Dans l'état SautMur, une transition de fourche permet de distinguer les sous-états du cheval et du cavalier. Le cheval est d'abord dans le sous-état Approche puis il se soulève lorsqu'il se situe dans le sous-état Enlevé
  - Enfin, il passe dans le sous-état Plané
  - Le cavalier reste dans le sous-état PositionSaut pendant le saut
  - Une transition de synchronisation est franchie à la fin du saut



# Le diagramme de timing

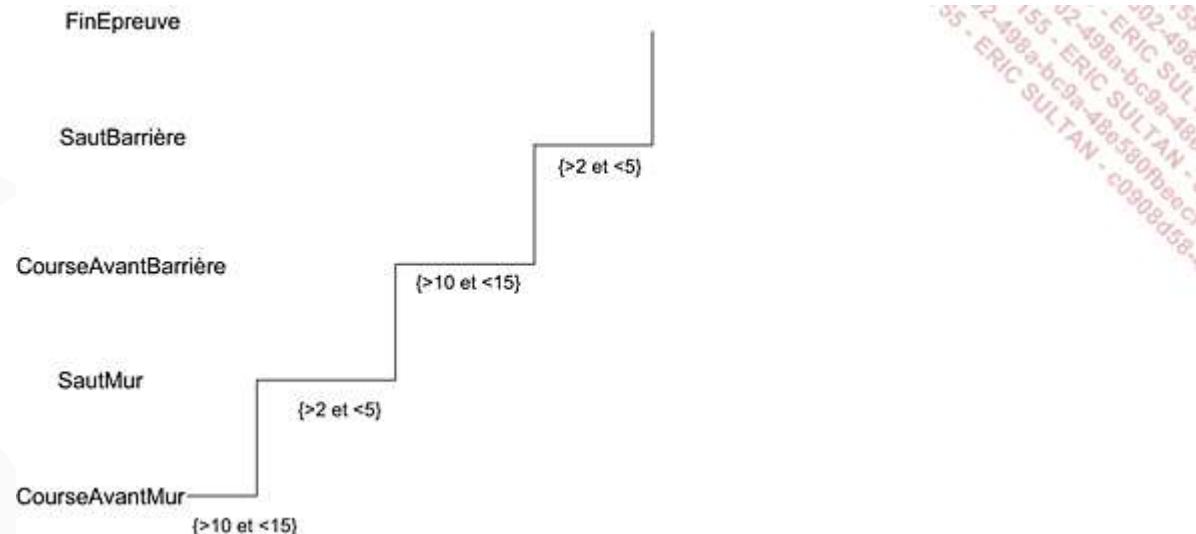
- Le diagramme de timing est introduit en UML 2 pour montrer les changements d'état d'un objet quand ceux-ci dépendent exclusivement du temps
- Le diagramme indique alors la durée minimale et/ou maximale de chaque état à l'aide de contraintes temporelles



# Le diagramme de timing

## □ Exemple :

- *Dans un concours d'obstacles, un cavalier doit s'imposer un temps maximum pour réaliser l'ensemble de l'épreuve, sinon il est éliminé.*
- *Il décompose lui-même ce temps sur chaque épreuve afin d'être sûr de réussir l'épreuve*



# Conclusion

- Le diagramme d'états-transitions décrit le cycle de vie des objets chargés d'assurer la dynamique du système
- Cette description du cycle de vie est réalisée séparément pour chacun de ces objets
- Cette modélisation est très importante pour s'assurer que les objets puissent répondre aux interactions décrites dans les diagrammes de séquence et de communication
- La modélisation de la dynamique

# **LA MODÉLISATION DES ACTIVITÉS**

# Introduction

- Le diagramme d'activités est basé sur le diagramme d'états-transitions déjà
- Il s'agit d'une forme spécifique du diagramme d'états-transitions dans lequel chaque état est associé à une activité et toutes les transitions sont automatiques
- Les transitions sont appelées enchaînements dans ce diagramme

# Introduction

- Le diagramme d'activités a ensuite été étendu pour décrire les activités de plusieurs objets
- Les enchaînements entre les activités de différents objets peuvent ainsi être représentés, ce qui n'est pas possible avec le diagramme d'états-transitions
- Nous verrons, pour chaque activité, comment désigner l'objet qui en est responsable grâce à la notion de travée
- Le diagramme d'activités offre des alternatives grâce aux conditions de garde
- Il peut également contenir des enchaînements de type fourche et synchronisation pour gérer des activités parallèles

# Les activités et les enchaînements d'activité

## □ Les activités :

- Une activité est une série d'actions
- Une action consiste à affecter une valeur à un attribut, créer ou détruire un objet, effectuer une opération, envoyer un signal à un autre objet ou à soi-même, etc...

Activité

## □ Exemple :

- *Nous reprenons l'exemple sur l'achat d'une jument.*
- *Le choix de la jument comme la vérification des vaccinations sont des exemples d'activité*

# Les activités et les enchaînements d'activité

- L'activité initiale est la première qui est exécutée. Elle est représentée par un point noir



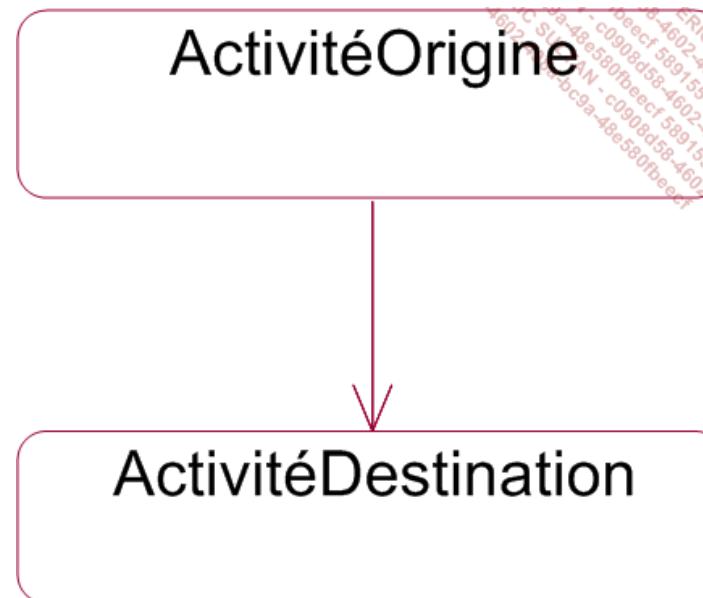
- Une activité finale représente la fin de l'exécution de l'ensemble des activités d'un diagramme
- Elle n'est pas forcément unique et n'est pas obligatoire.
- Une activité finale est représentée par un point noir entouré d'un cercle



# Les activités et les enchaînements d'activité

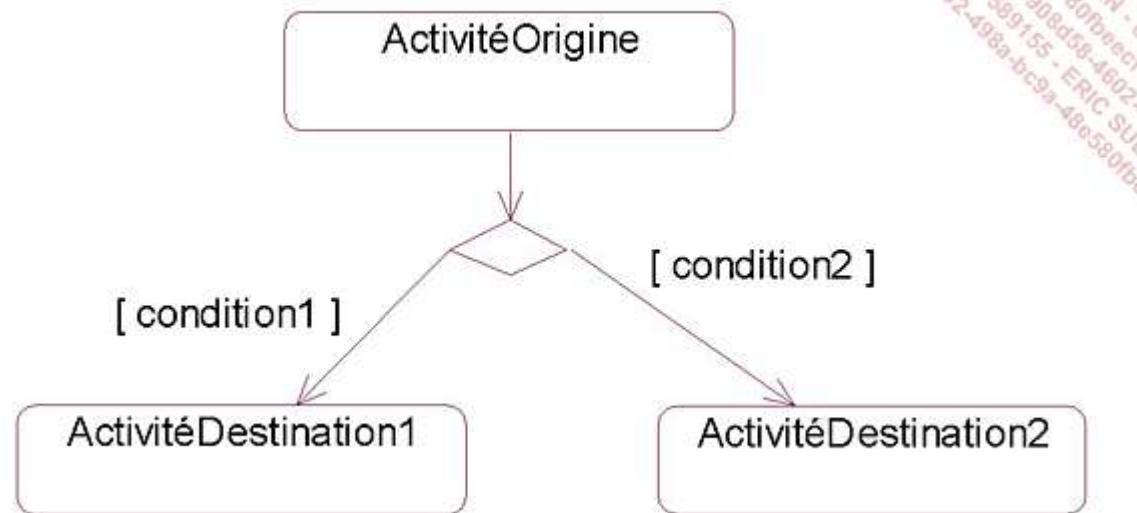
## □ Les enchaînements d'activités :

- Un enchaînement d'activités est un lien orienté entre deux activités
- Il est franchi dès que l'activité d'origine est terminée
- Son franchissement conduit à l'enclenchement de l'activité de destination



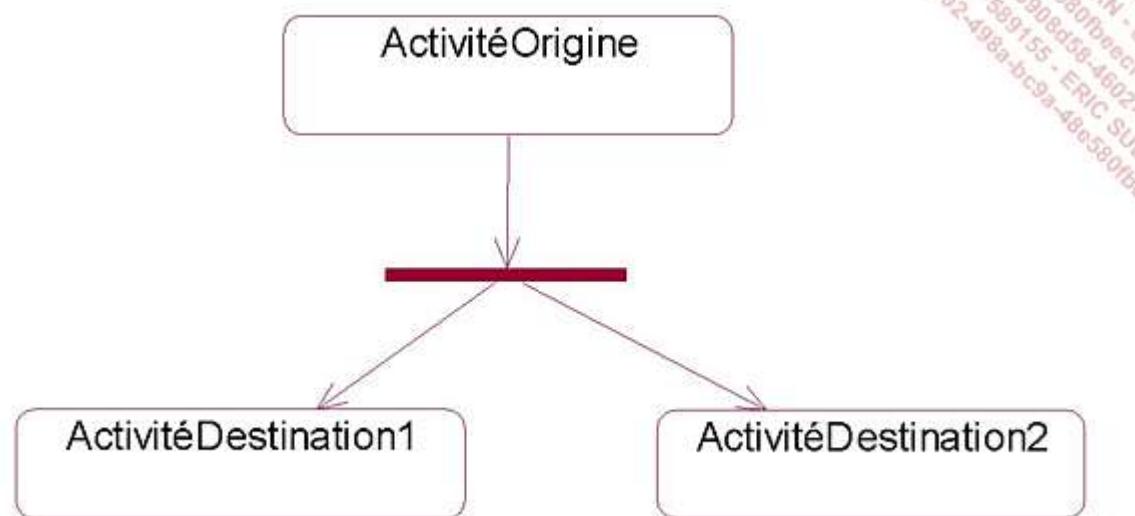
# Les activités et les enchaînements d'activité

- Un enchaînement d'activités peut également être une alternative
- Chaque branche de l'alternative est dotée d'une condition de garde exclusive des autres conditions



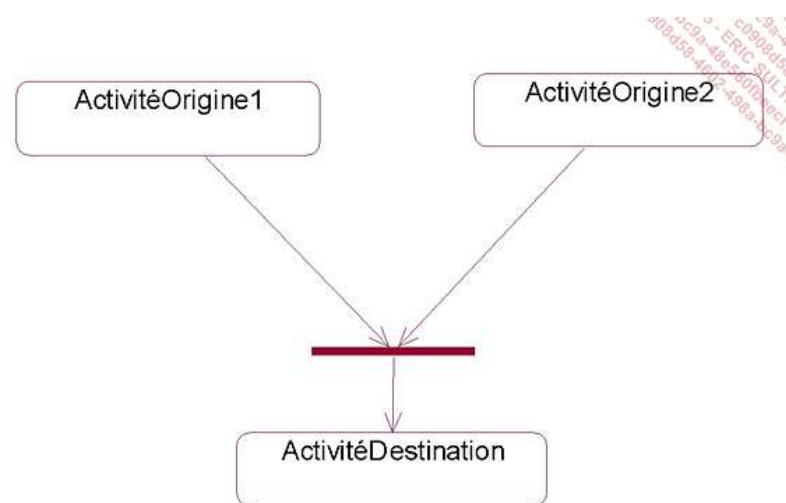
# Les activités et les enchaînements d'activité

- Un enchaînement d'activités de type *fourche* possède également plusieurs activités de destination
- Dès qu'il est franchi, toutes les activités de destination sont enclenchées en parallèle



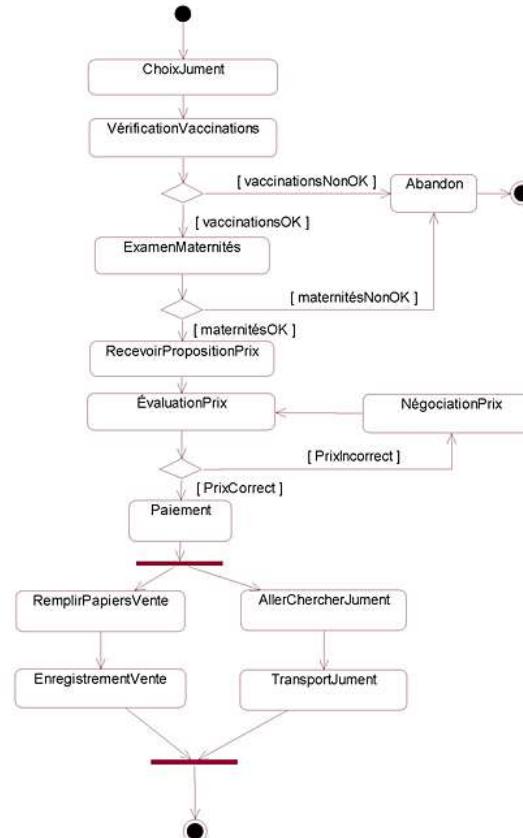
# Les activités et les enchaînements d'activité

- Un enchaînement d'activités de type *synchronisation* possède plusieurs activités d'origine et une seule activité de destination
- Il faut que toutes les activités d'origine soient terminées pour qu'il soit franchi et que l'activité de destination soit enclenchée



# Les activités et les enchaînements d'activité

- Exemple :
  - Nous reprenons l'exemple de l'achat d'une jument
  - Le traitement des papiers et le transport de la jument sont traités en parallèle. En effet, l'acheteur peut très bien réaliser ces deux activités en même temps
  - Les conditions de garde expriment les différentes alternatives. Il convient aussi de noter la présence de deux activités finales, l'une correspondant à l'abandon et l'autre à un achat réussi

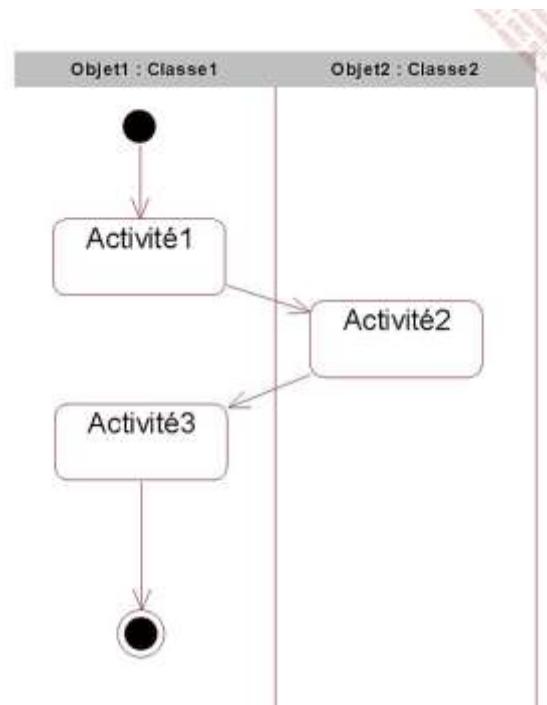


# Les couloirs

- À la différence du diagramme d'états-transitions, le diagramme d'activités peut représenter les activités réalisées par plusieurs objets avec leurs enchaînements
- Pour cela, le diagramme est divisé en couloirs.
- À chaque couloir correspond l'objet responsable de la réalisation de toutes les activités contenues dans ce couloir

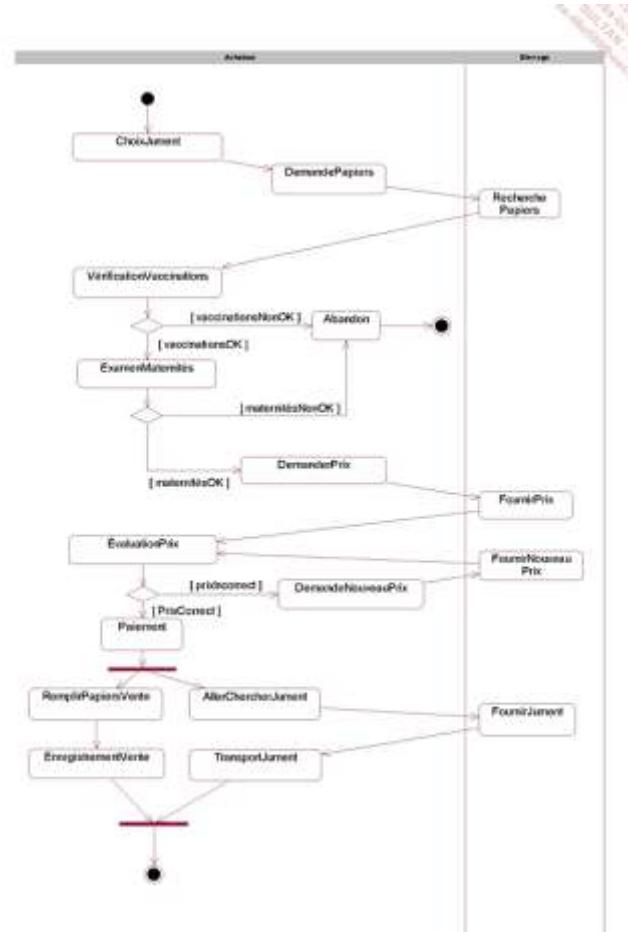
# Les couloirs

- Un enchaînement peut couper la ligne de séparation de deux couloirs pour montrer un changement d'objet entre l'activité d'origine et celle de destination



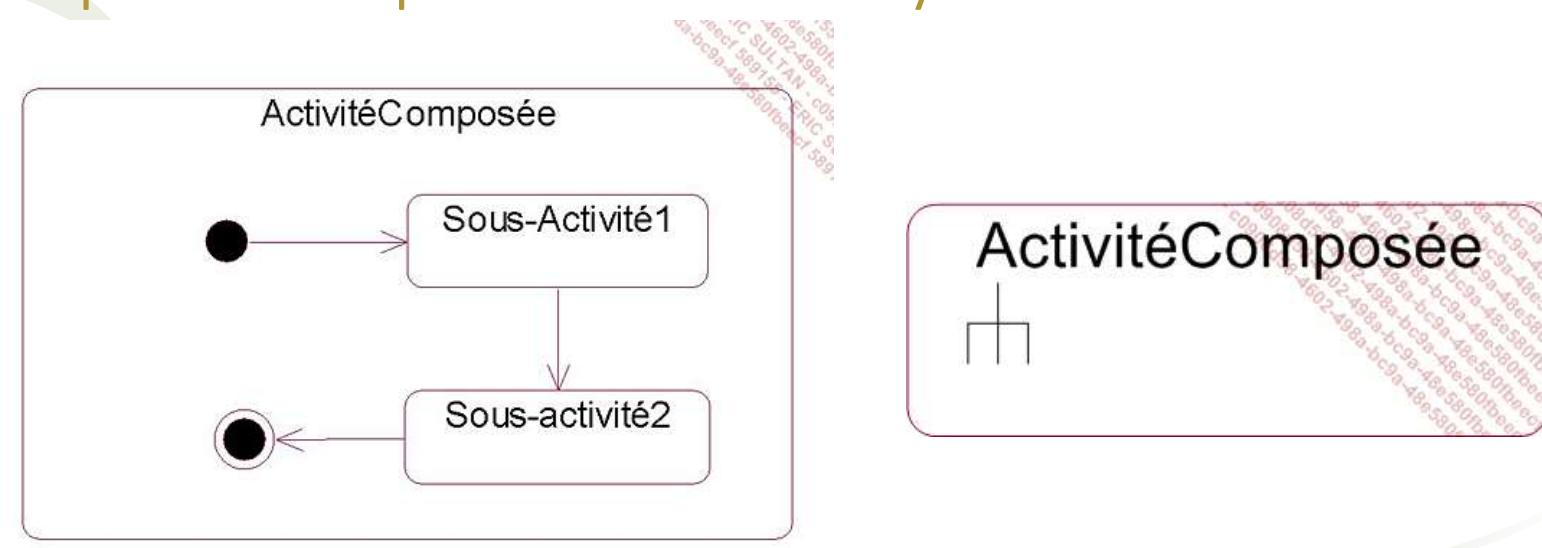
# Les couloirs

- Exemple :
  - Illustrons l'exemple de l'achat d'une jument où les activités relatives à l'acheteur et à l'élevage vendeur sont décrites



# Les activités composées

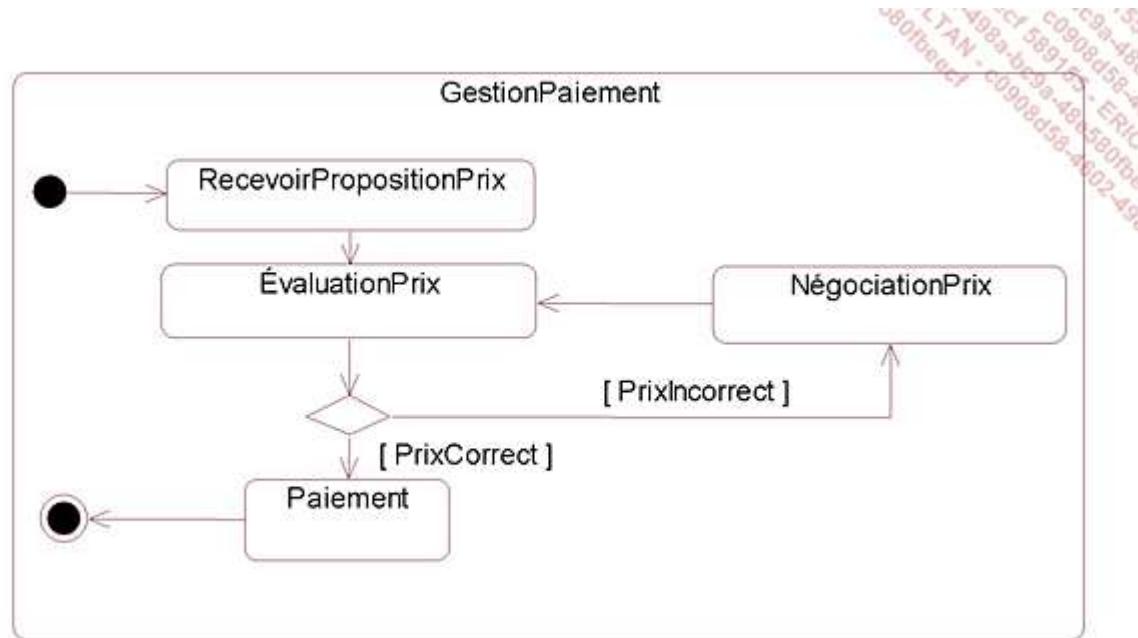
- Une activité peut être composée d'autres activités
- Dans ce cas, un diagramme d'activités spécifique en décrit la composition en sous-activités
- Dans les diagrammes où elle est présente, une activité composée est représentée avec un symbole de fourche



# Les activités composées

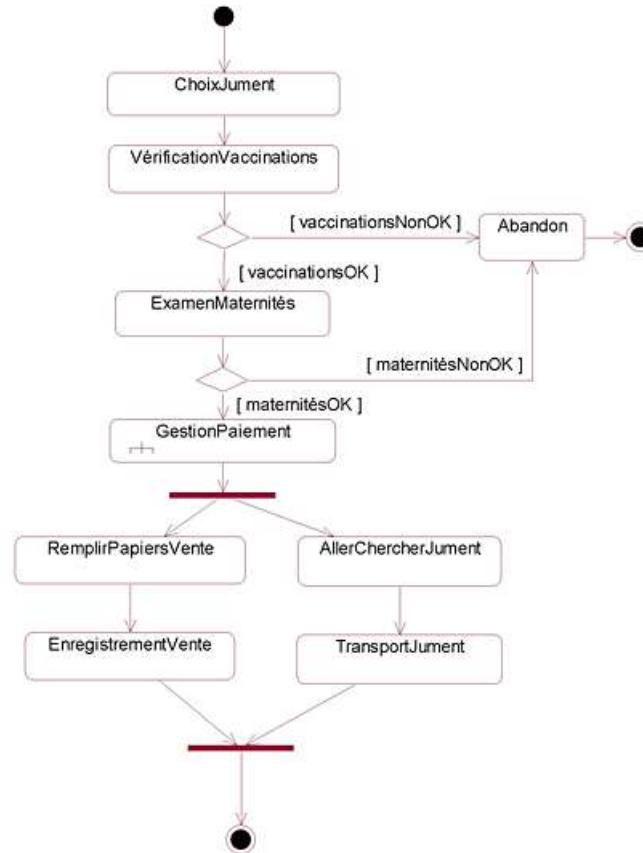
## □ Exemple :

- *Illustrons la gestion du paiement d'une jument en intégrant la négociation du prix.*



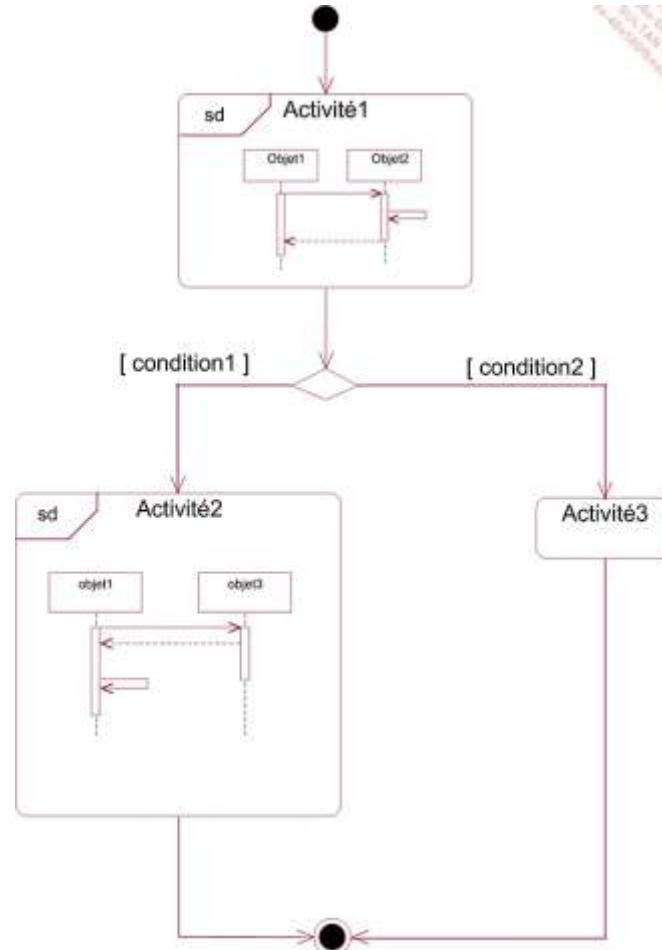
# Les activités composées

## □ Exemple (suite) :



# Le diagramme de vue d'ensemble des interactions

- Le diagramme de vue d'ensemble des interactions est spécifique à UML 2
- Il s'agit d'un diagramme d'activités où chaque activité peut être décrite par un diagramme de séquence



# Conclusion

- Le diagramme d'activités représente les activités que réalisent un ou plusieurs objets
- Il peut correspondre à la description en détail d'une activité du diagramme d'états-transitions, à la description d'une méthode
- Il peut également décrire l'activité d'un système ou d'un sous-système en assignant les responsabilités à chaque acteur
- Le diagramme d'activités constitue aussi un bon choix pour décrire un cas d'utilisation

# **LA MODÉLISATION DE L'ARCHITECTURE SYSTÈME**

# Introduction

- Dans ce chapitre, nous aborderons les possibilités offertes par UML pour modéliser l'architecture du système
- Cette modélisation se décline sous deux aspects :
  - la modélisation de l'architecture logicielle et sa structuration en composants
  - la modélisation de l'architecture matérielle et la répartition physique des logiciels
- Nous étudierons la notion de composant logiciel
- Un composant est une boîte noire qui offre des services logiciels
- Ces services sont décrits par une ou plusieurs interfaces du composant

# Introduction

- Rappelons qu'une interface est une classe abstraite ne contenant que des signatures de méthodes
- La signature d'une méthode est composée de son nom et de ses paramètres
- Un composant peut également dépendre d'autres composants pour réaliser les services qu'il offre
- Cette dépendance s'exprime sous la forme d'une interface requise qui décrit les services attendus

# Introduction

- La modélisation des composants et de leurs relations est décrite par le diagramme des composants
- La modélisation de l'architecture matérielle décrit les nœuds et leurs liaisons
- Elle inclut la localisation des éléments logiciels au sein des nœuds, sous leur forme physique appelée *artefact*
- Sa description est donnée par le diagramme de déploiement

# Le diagramme des composants

## □ Les composants :

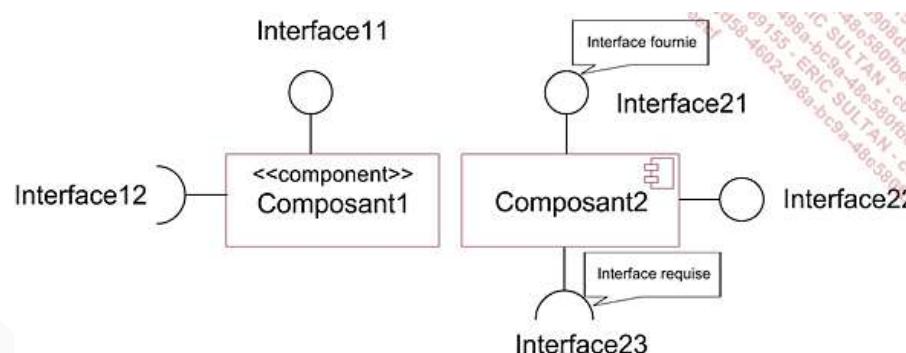
- Un composant est une unité logicielle offrant des services au travers d'une ou de plusieurs interfaces
- C'est une boîte noire dont le contenu n'intéresse pas ses clients
- Il est totalement encapsulé
- Il existe plusieurs technologies de composant :
  - les composants COM et .NET
  - les composants Java : JavaBeans et Enterprise JavaBeans
  - ...

# Le diagramme des composants

- Un composant peut dépendre d'autres composants pour réaliser ses opérations internes
- Il est alors le client de ces autres composants
- Comme tout client d'un composant, il n'en connaît pas la structure interne
- Il ne dépend donc que de la ou des interfaces des composants dont il est client
- Ces interfaces sont appelées les **interfaces requises** du composant *client*
- Les interfaces qui décrivent les services offerts par un composant sont appelées ses **interfaces fournies**

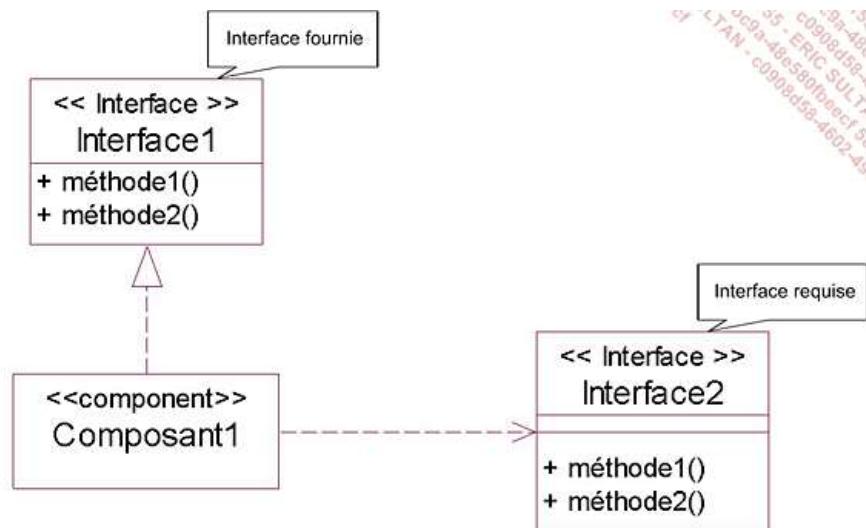
# Le diagramme des composants

- La notation graphique d'une interface fournie est identique à celle que nous avons présentée lors de la modélisation des objets
- Une interface requise est représentée par un demi-cercle
- Un composant est représenté dans un rectangle avec le stéréotype «component»
- Ce stéréotype peut être remplacé par le logo du composant.



# Le diagramme des composants

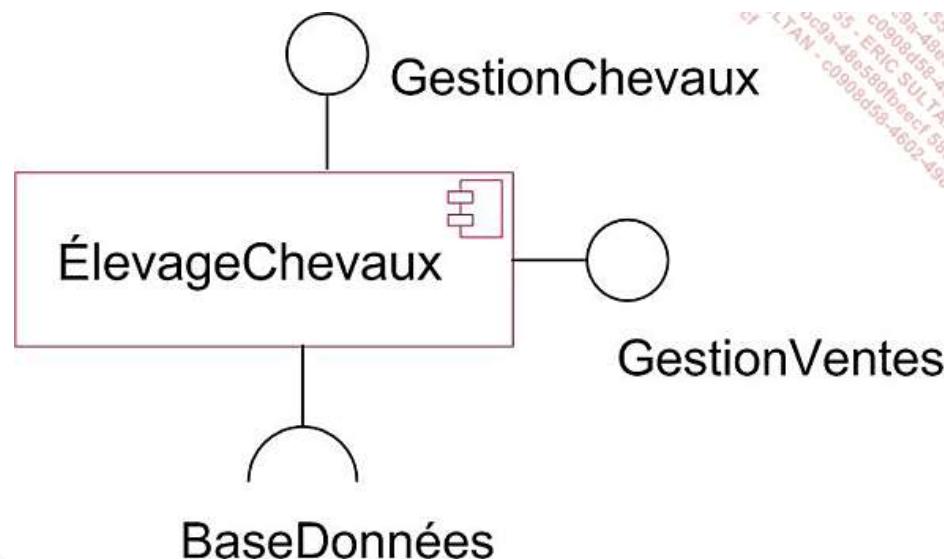
- Il est également possible d'utiliser la relation de réalisation pour les interfaces fournies et la relation de dépendance pour les interfaces requises
- Celle-ci présente l'avantage de détailler les signatures de méthode contenues dans les interfaces



# Le diagramme des composants

## □ Exemple :

- *Un composant de gestion d'un élevage de chevaux fournit une interface pour la gestion des chevaux et une interface pour la gestion des ventes*
- *Par ailleurs, il requiert un composant de base de données*



# Le diagramme des composants

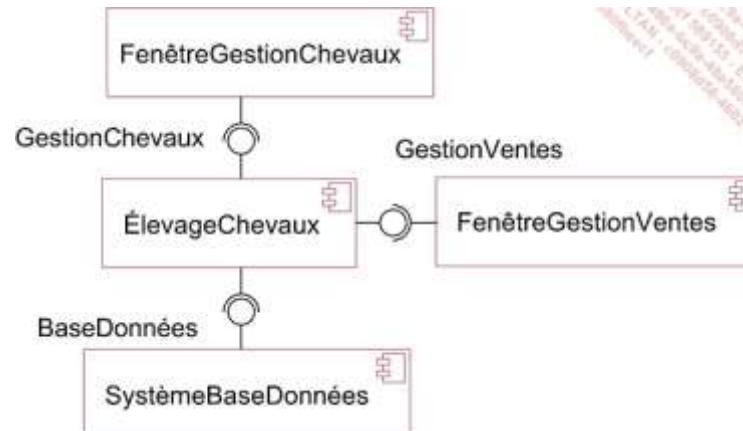
## □ L'architecture logicielle par composants :

- Dans l'approche par objets, l'architecture logicielle d'un système est construite par un assemblage de composants liés par des interfaces fournies et des interfaces requises
- Le diagramme des composants décrit cette architecture

# Le diagramme des composants

## □ Exemple :

- *Le système d'information d'un élevage de chevaux est réalisé par assemblage de composants*
- *Les composants FenêtreGestionChevaux et FenêtreGestionVentes gèrent respectivement à l'écran une fenêtre consacrée à la gestion des chevaux et à la gestion des ventes de chevaux*
- *Le composant SystèmeBaseDonnées est un système de base de données*

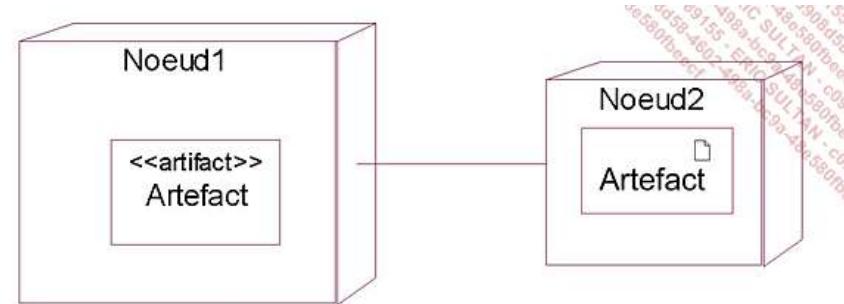


# Le diagramme de déploiement

- Le diagramme de déploiement décrit l'architecture physique du système
- Celui-ci est composé de nœuds
- Un nœud est une unité matérielle capable de recevoir et d'exécuter du logiciel
- La plupart des nœuds sont des ordinateurs
- Les liaisons physiques entre nœuds peuvent également être décrites dans le diagramme de déploiement
- Elles correspondent aux branches du réseau

# Le diagramme de déploiement

- ❑ Les nœuds contiennent des logiciels sous leur forme physique
- ❑ Cette dernière est nommée artefact
- ❑ Un fichier exécutable, une bibliothèque partagée ou un script sont des exemples de forme physique de logiciel
- ❑ Les composants qui constituent l'architecture logicielle du système sont représentés dans le diagramme de déploiement par un artefact qui est souvent un exécutable ou une bibliothèque partagée



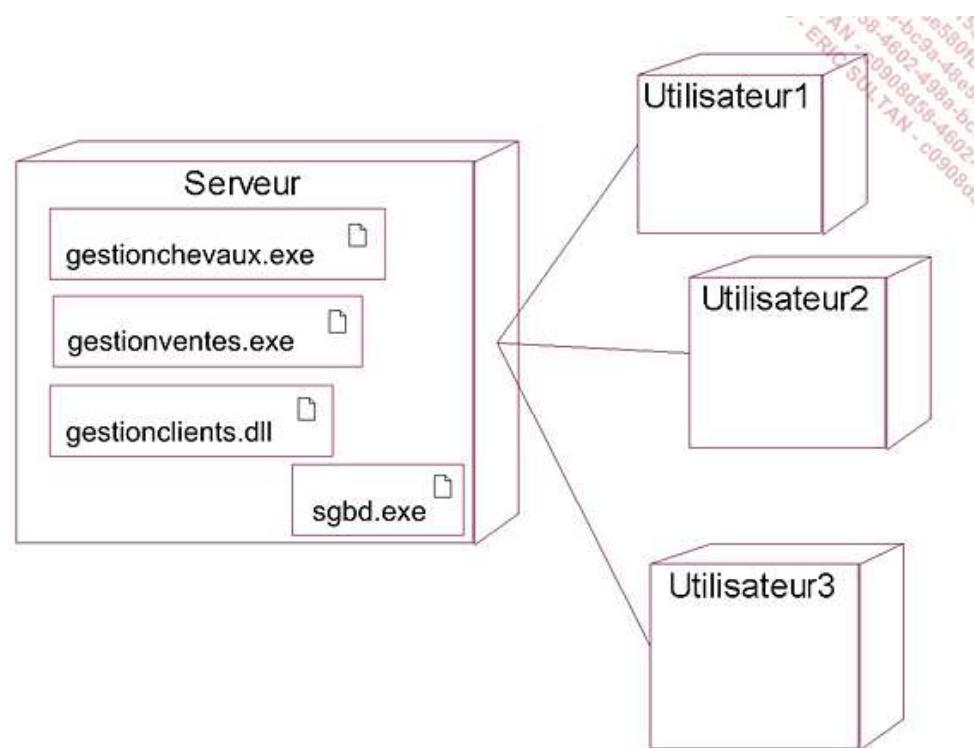
# Le diagramme de déploiement

## □ Exemple :

- *Illustrons l'architecture matérielle du système d'information d'un élevage de chevaux*
- *Cette architecture est basée sur un serveur et trois postes clients.*
- *Ces derniers sont connectés au serveur par des liens directs*
- *Le serveur contient plusieurs artefacts :*
  - *un exécutable (.exe), forme physique du composant de gestion de la base de données*
  - *un deuxième exécutable chargé de la gestion des chevaux*
  - *un troisième exécutable chargé de la gestion des ventes*
  - *une bibliothèque partagée (.dll) de gestion des machines des différents utilisateurs*

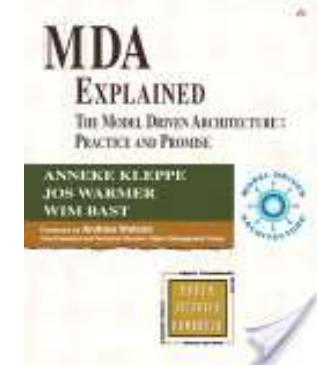
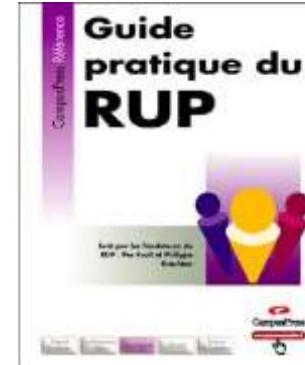
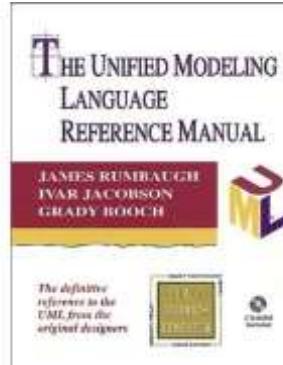
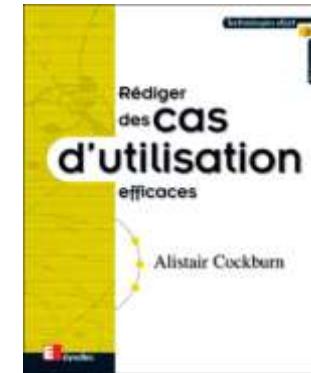
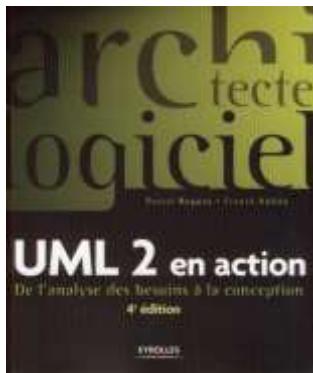
# Le diagramme de déploiement

## □ Exemple (suite) :

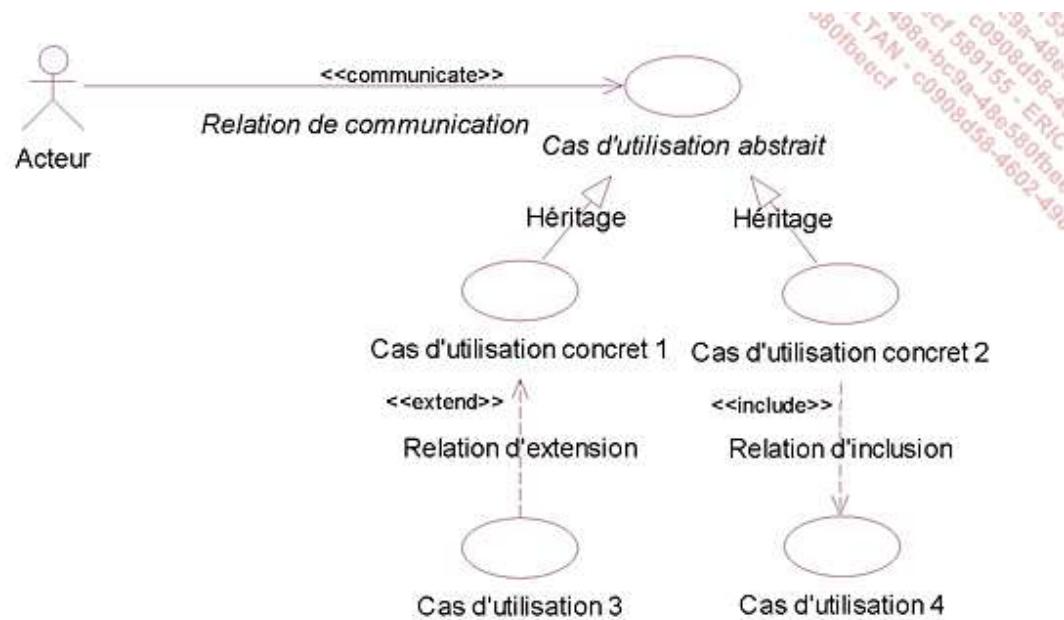


# BIBLIOGRAPHIE

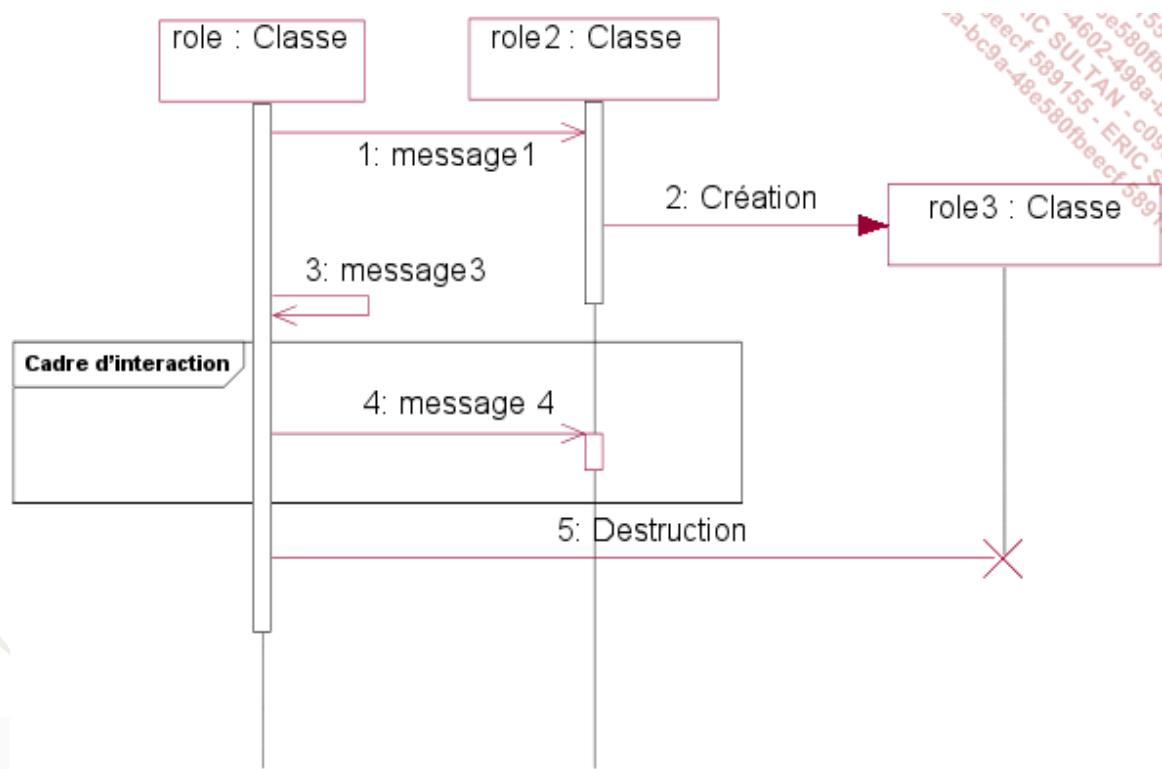
# Quelques livres



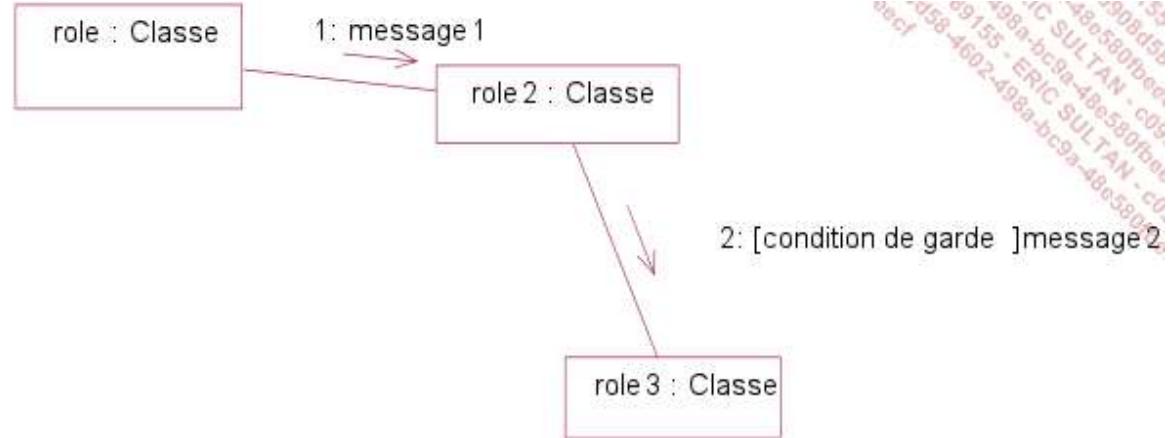
# **NOTATION GRAPHIQUE**



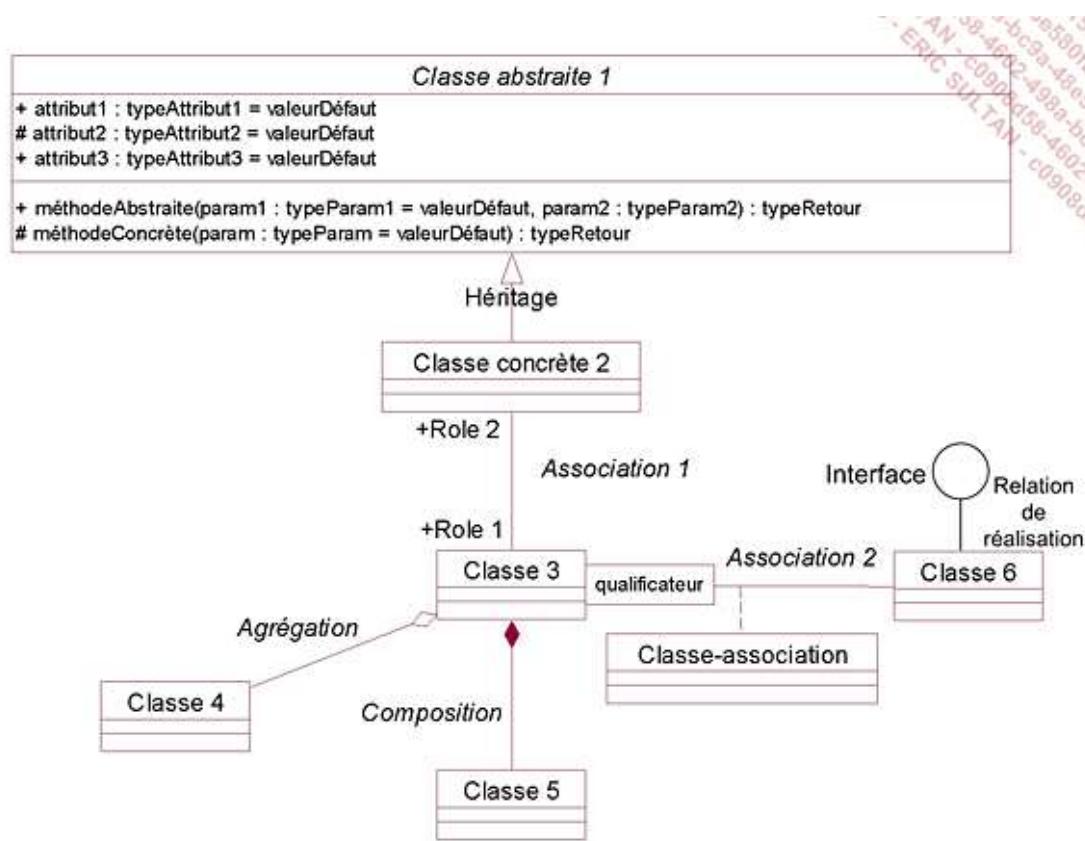
## Diagramme de cas d'utilisation



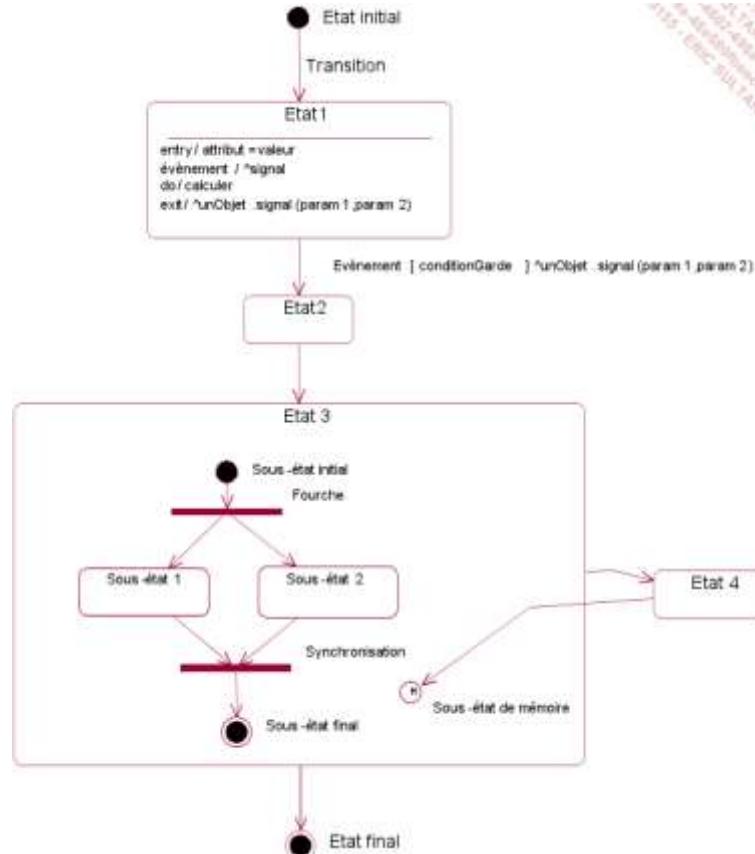
## Diagramme de séquence



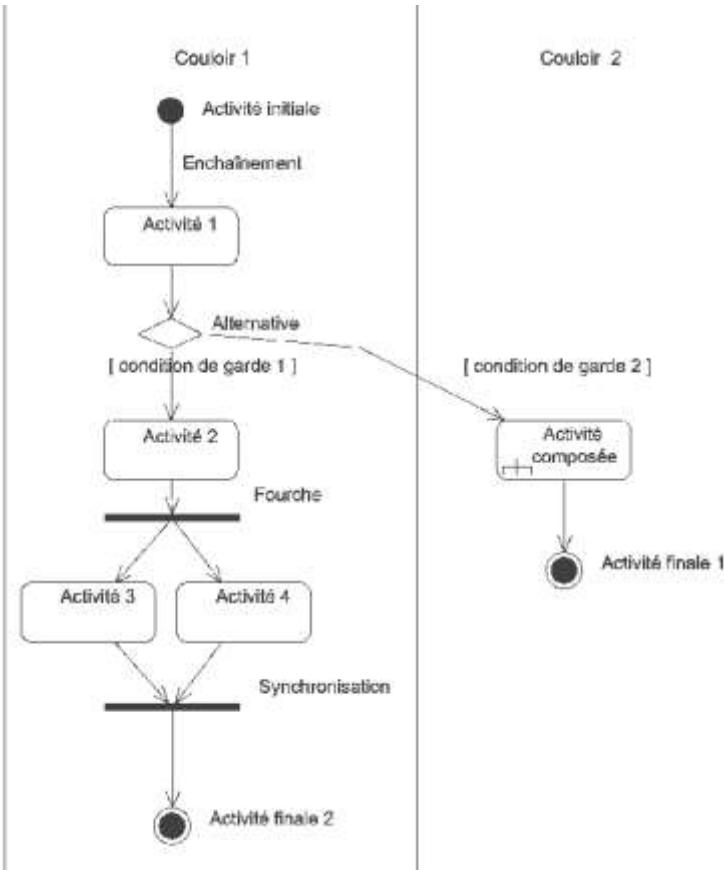
## Diagramme de communication



## Diagramme de classes



## Diagramme d'états-transitions



## Diagramme d'activités

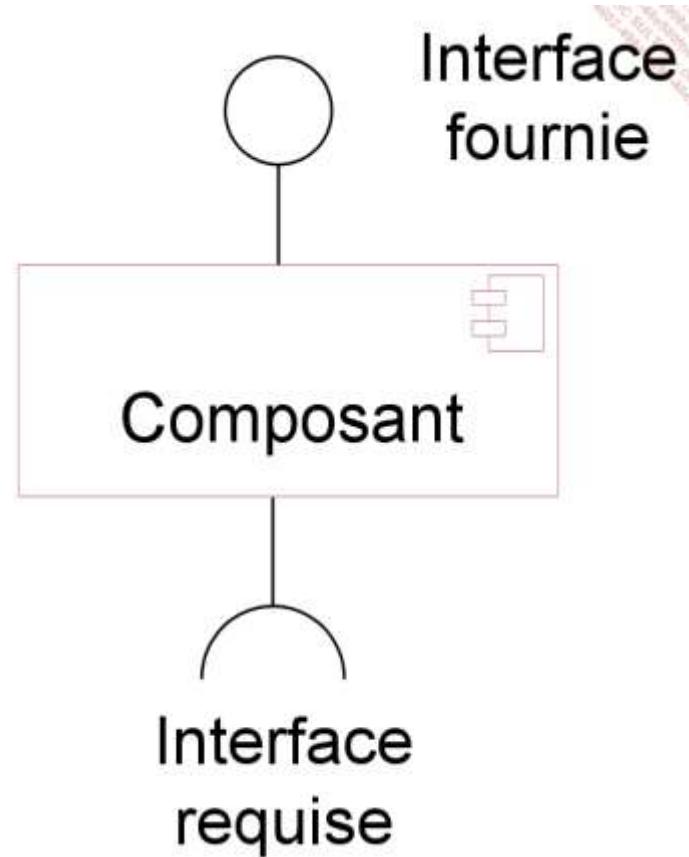
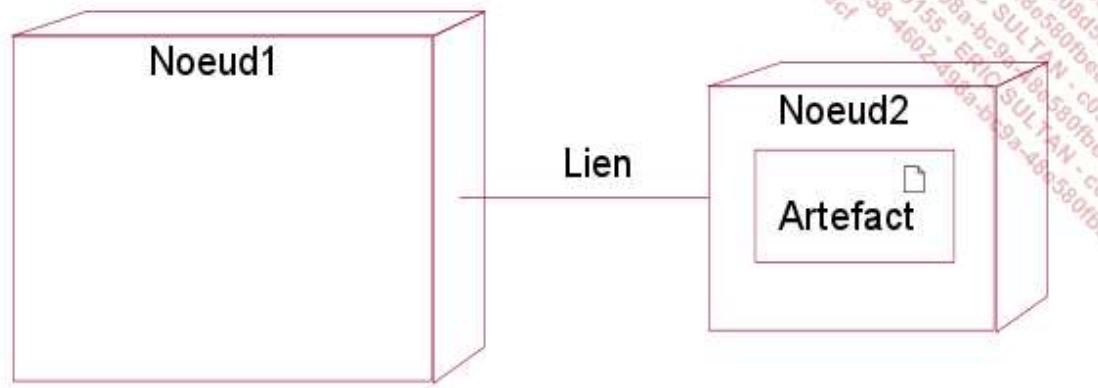


Diagramme des composants



## Diagramme de déploiement