

# **Introduction**

## **à la Modélisation par Objets :**

### **A la découverte des objets**

### **et des classes**



*Isabelle BLASQUEZ*  
*@iblasquez*

*Janvier 2022*



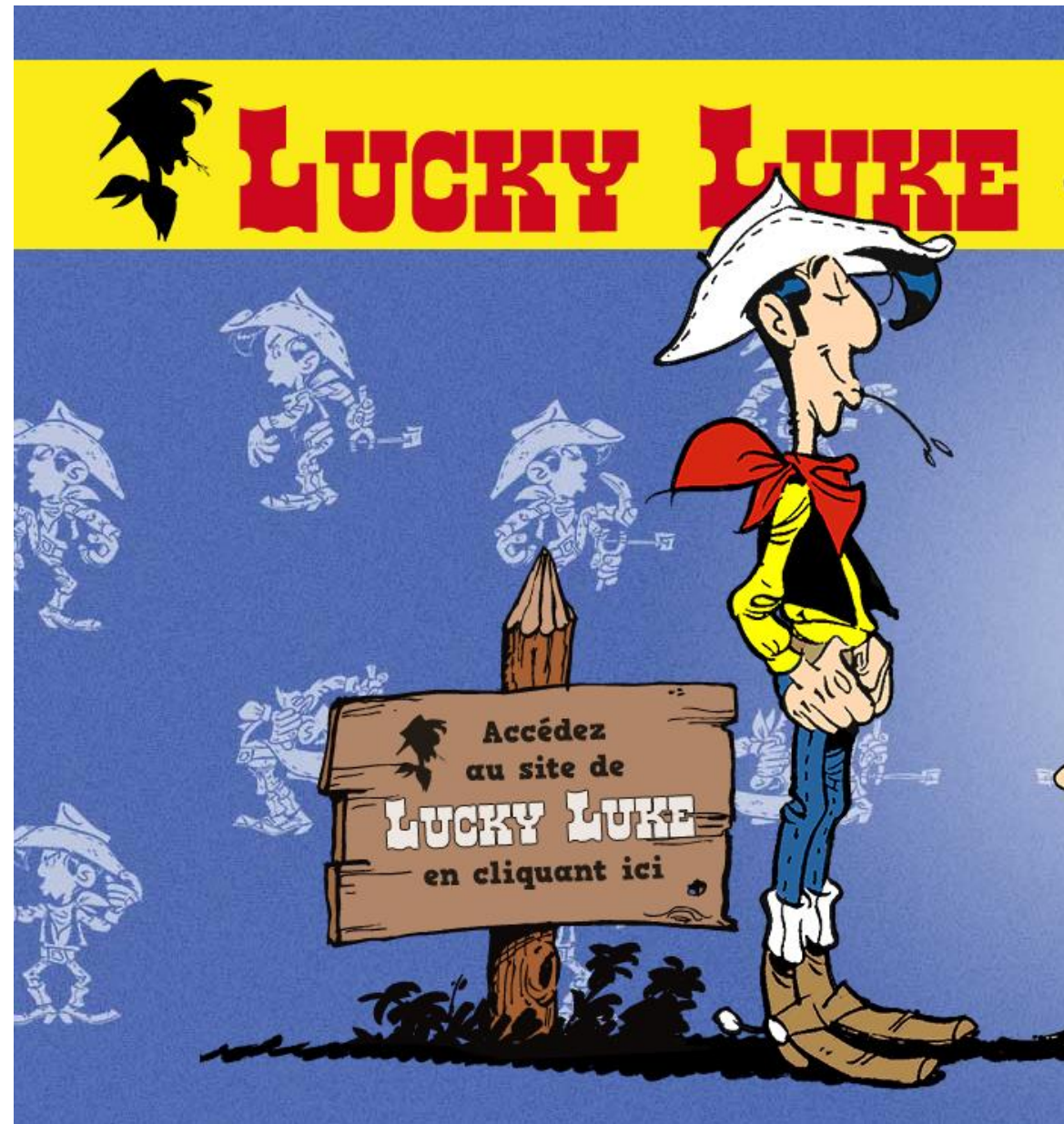
**Isabelle Blasquez**

@iblasquez

Associate Professor in Software Engineering [#SoftwareCraftsmanship](#) [#iutagile](#)  
[#limouzicodev](#) [@duchessfr](#) [@CodeWeekEU](#) [@MuseomixLIM](#) [@aperscope87](#)

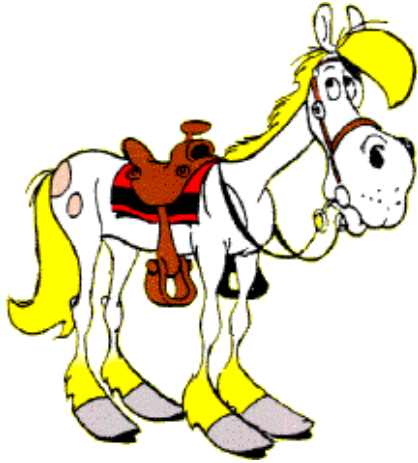
# A la découverte des *Objets*

*(avec les personnages  
de Lucky Luke)*



Extrait de : <http://www.lucky-luke.com/>

# Présentation simplifiée du personnage du cheval de Lucky Luke



Jolly Jumper est le cheval de Lucky Luke :  
il accompagne Lucky Luke dans toutes ses aventures.  
On dit de lui que c'est « le cheval le plus rapide et le  
plus intelligent de l'Ouest »

Jolly Jumper a été créé en 1946.  
Jolly Jumper est un étalon de robe blanche,  
de la race appaloosa.



Jolly Jumper est un cheval savant  
capable de parler.  
Mais il est également très susceptible. Ainsi si Lucky  
Luke a le malheur de se moquer de lui, Jolly Jumper  
peut bouder pendant des heures : il sera par exemple  
capable de galoper pendant des heures sans lui  
adresser la parole.



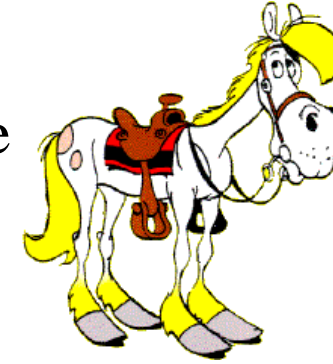
# Qu'apprenons-nous sur le cheval de Lucky Luke ?

*nom : Jumper*  
*prénom : Jolly*

*propriétaire : Lucky Luke*

Jolly Jumper est le cheval de Lucky Luke : il accompagne Lucky Luke dans toutes ses aventures.

On dit de lui que c'est « le cheval le plus rapide et le plus intelligent de l'Ouest »



*date de création : 1946*

Jolly Jumper a été créé en 1946.

*race: appaloosa*

Jolly Jumper est un étalon de robe blanche, de la race appaloosa.

*couleur robe: blanche*

*Il peut parler*

Jolly Jumper est un cheval savant capable de parler.

Mais il est également très susceptible. Ainsi si Lucky Luke a le malheur de se moquer de lui, Jolly Jumper peut boudier pendant des heures :

il sera par exemple capable de galoper pendant des heures sans lui adresser la parole.

*Il peut boudier*

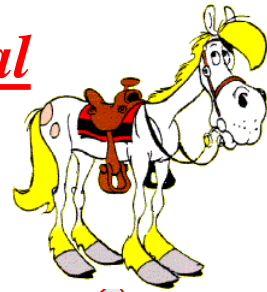
*Il peut galoper*



# Comment organiser les informations concernant le cheval de Lucky Luke ?

Ces informations permettent :

→ d'une part de **décrire le cheval**  
à partir de  
ce que l'on **SAIT** sur lui  
(aspect *statique*)



*nom : Jumper*  
*prénom : Jolly*

*propriétaire : Lucky Luke*

*date de création : 1946*

*race: appaloosa*

*couleur robe: blanche*

→ d'autre part **de connaître le(s) différent(s) comportement(s) possible(s)**  
du cheval en répondant à la question «Que **FAIT** le cheval ?»  
(aspect *dynamique*)

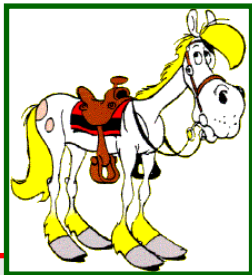
*Il peut galoper*

*Il peut boudier*

*Il peut parler*

# Regroupons ces informations dans une même « boîte »

Regroupons ces informations  
dans une même boîte  
« à notre façon »....



*nom : Jumper*  
*prénom : Jolly*  
*date de création : 1946*  
*propriétaire : Lucky Luke*  
*race : appaloosa*  
*Couleur robe : blanche*

*Il peut galoper*  
*Il peut bouter*  
*Il peut parler*

... puis adoptons le **formalisme UML**  
pour modéliser le cheval de Lucky Luke



*En UML, pour pouvoir désigner et  
manipuler un objet, on doit lui  
donner une identité*

jollyJumperleChevalDeLuckyLuke

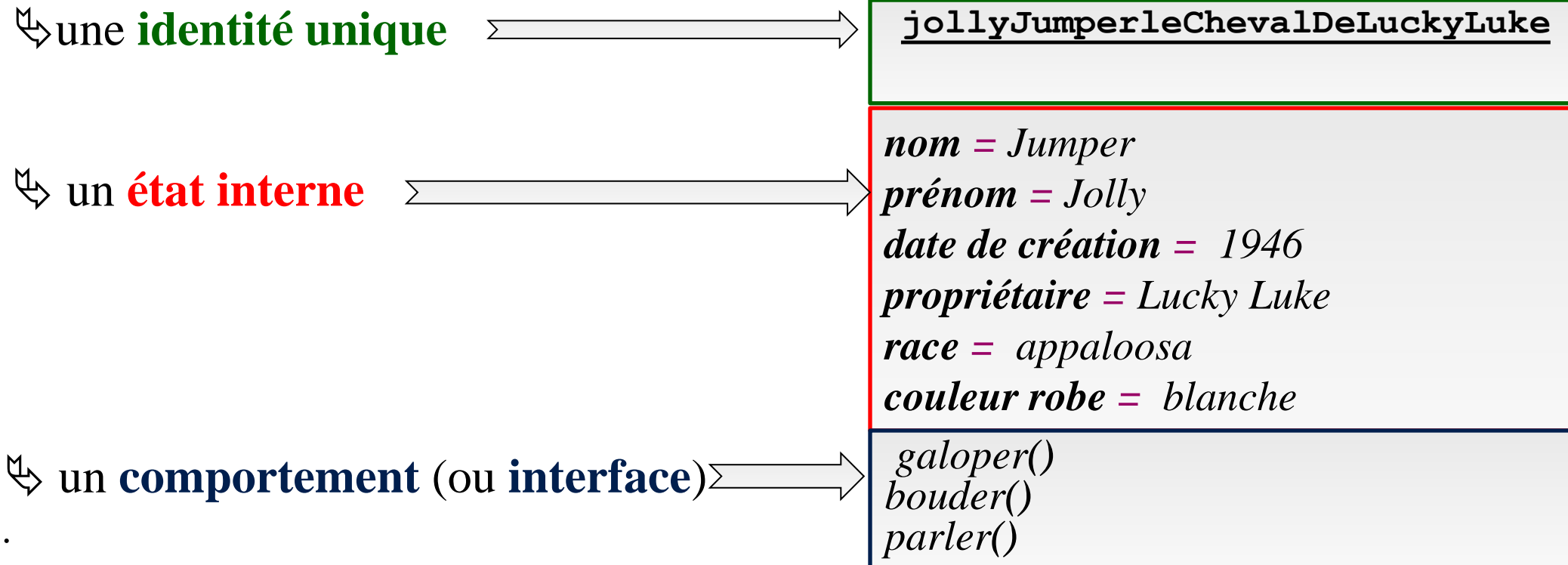
*nom = Jumper*  
*prénom = Jolly*  
*date de création = 1946*  
*propriétaire = Lucky Luke*  
*race = appaloosa*  
*couleur robe = blanche*

*galoper()*  
*bouter()*  
*parler()*

# Objet : Définition

Toute entité identifiable, concrète ou abstraite  
peut être définie comme objet

Un objet possède **3 caractéristiques** :





# Intéressons-nous maintenant au personnage de Lucky Luke

*Il peut tirer au pistolet*

*nom : Luke  
prénom : Lucky*

*sexe : masculin*

Lucky Luke est connu pour être « *l'homme qui tire au pistolet plus vite que son ombre* ».

Lucky Luke a été créé en 1946.

*date de création :  
1946*

Lucky Luke mesure 1 mètre 75.

*taille : 1 m 75*

*profession : cow-boy*

*Il peut monter à cheval*

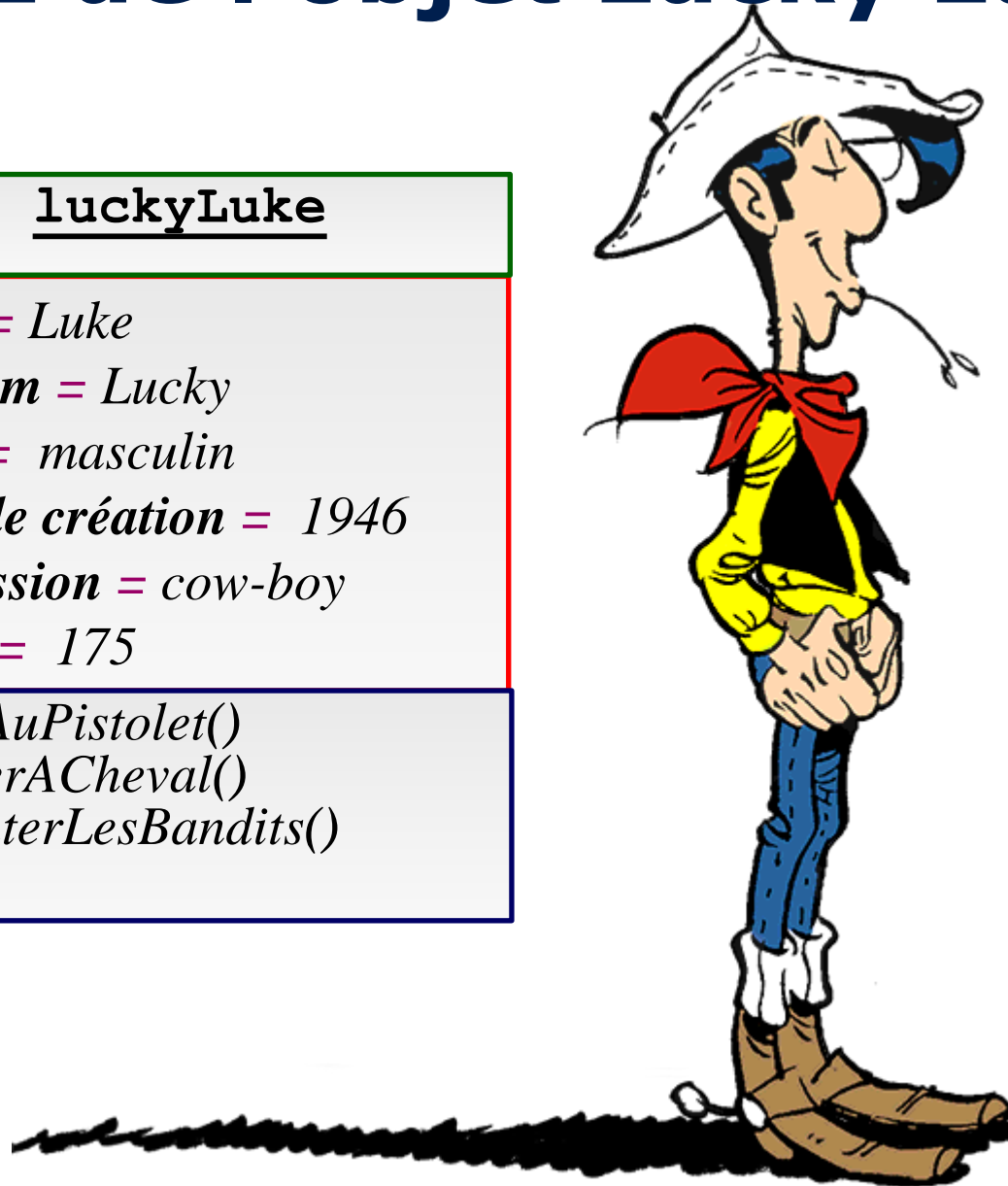
Comme tout cow-boy, Lucky Luke sait monter à cheval

Il n'hésite pas à affronter les bandits pour que la loi continue à régner dans l'Ouest américain.

*Il peut affronter  
les bandits*



# Notation UML de l'objet Lucky Luke



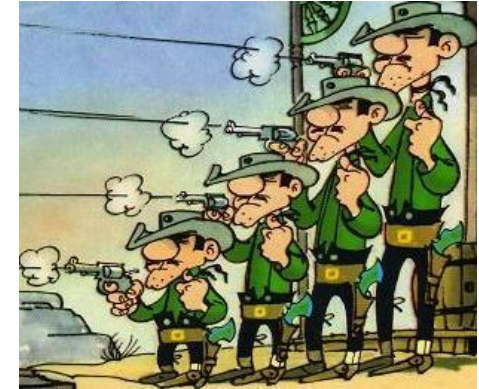
# Et les Daltons ?

joeDalton

*nom = Dalton*  
*prénom = Joe*  
*sexe = masculin*  
*date de création = 1954*  
*profession = bandit*  
*taille = 150*

*creuserUnTunnel()*  
*tirerAuPistolet()*

Les Daltons sont 4 bandits.  
Ils ont été créés en 1954.  
Du plus petit au plus grand,  
on trouve Joe (1,50 m),  
William (1,70 m.), Jack (1,80 m) et Averell (1,90 m)  
Lorsqu'ils sont en prison, les Daltons creusent  
souvent des tunnels.  
Ils savent également tirer au pistolet.



williamDalton

*nom = Dalton*  
*prénom = William*  
*sexe = masculin*  
*date de création = 1954*  
*profession = bandit*  
*taille = 170*

*creuserUnTunnel()*  
*tirerAuPistolet()*

jackDalton

*nom = Dalton*  
*prénom = Jack*  
*sexe = masculin*  
*date de création = 1954*  
*profession = bandit*  
*taille = 180*

*creuserUnTunnel()*  
*tirerAuPistolet()*

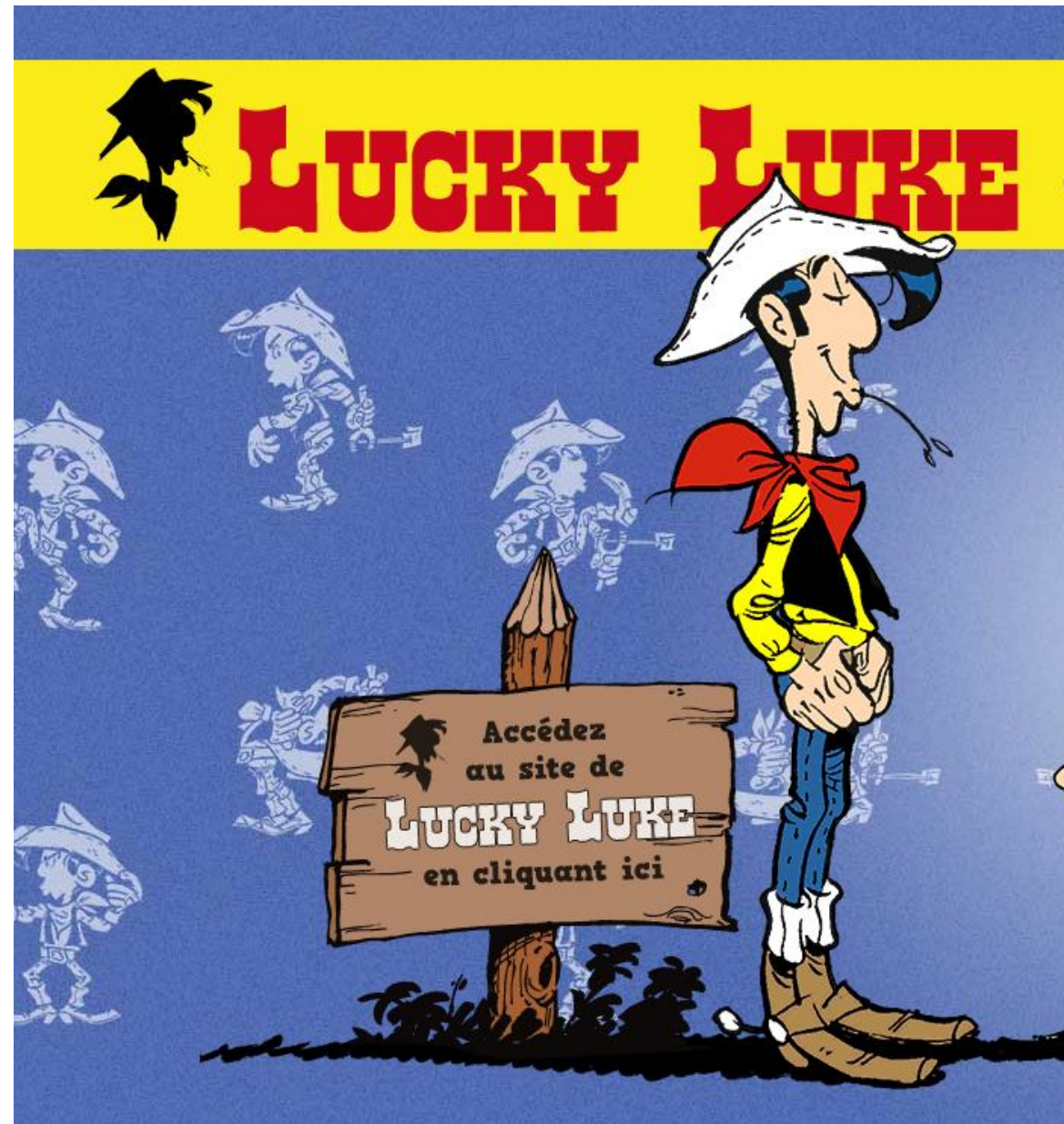
averellDalton

*nom = Dalton*  
*prénom = Averell*  
*sexe = masculin*  
*date de création = 1954*  
*profession = bandit*  
*taille = 190*

*creuserUnTunnel()*  
*tirerAuPistolet()*

# A la découverte des *Classes*

*(avec les personnages  
de Lucky Luke)*



Extrait de : <http://www.lucky-luke.com/>

# Ressemblances et différences entre les membres de la famille Daltons ?

<u>joeDalton</u>	<u>williamDalton</u>	<u>jackDalton</u>
<i>nom = Dalton</i> <i>prénom = Joe</i> <i>sexe = masculin</i> <i>date de création = 1954</i> <i>profession = bandit</i> <i>taille = 150</i>	<i>nom = Dalton</i> <i>prénom = William</i> <i>sexe = masculin</i> <i>date de création = 1954</i> <i>profession = bandit</i> <i>taille = 170</i>	<i>nom = Dalton</i> <i>prénom = Jack</i> <i>sexe = masculin</i> <i>date de création = 1954</i> <i>profession = bandit</i> <i>taille = 180</i>
<i>creuserUnTunnel()</i> <i>tirerAuPistolet()</i>	<i>creuserUnTunnel()</i> <i>tirerAuPistolet()</i>	<i>creuserUnTunnel()</i> <i>tirerAuPistolet()</i>

**Différences**

**Identité propre**

**Etat interne propre**

**Ressemblances**

**Comportement  
commun**

**averellDalton**

*nom = Dalton*  
*prénom = Averell*  
*sexe = masculin*  
*date de création = 1954*  
*profession = bandit*  
*taille = 190*

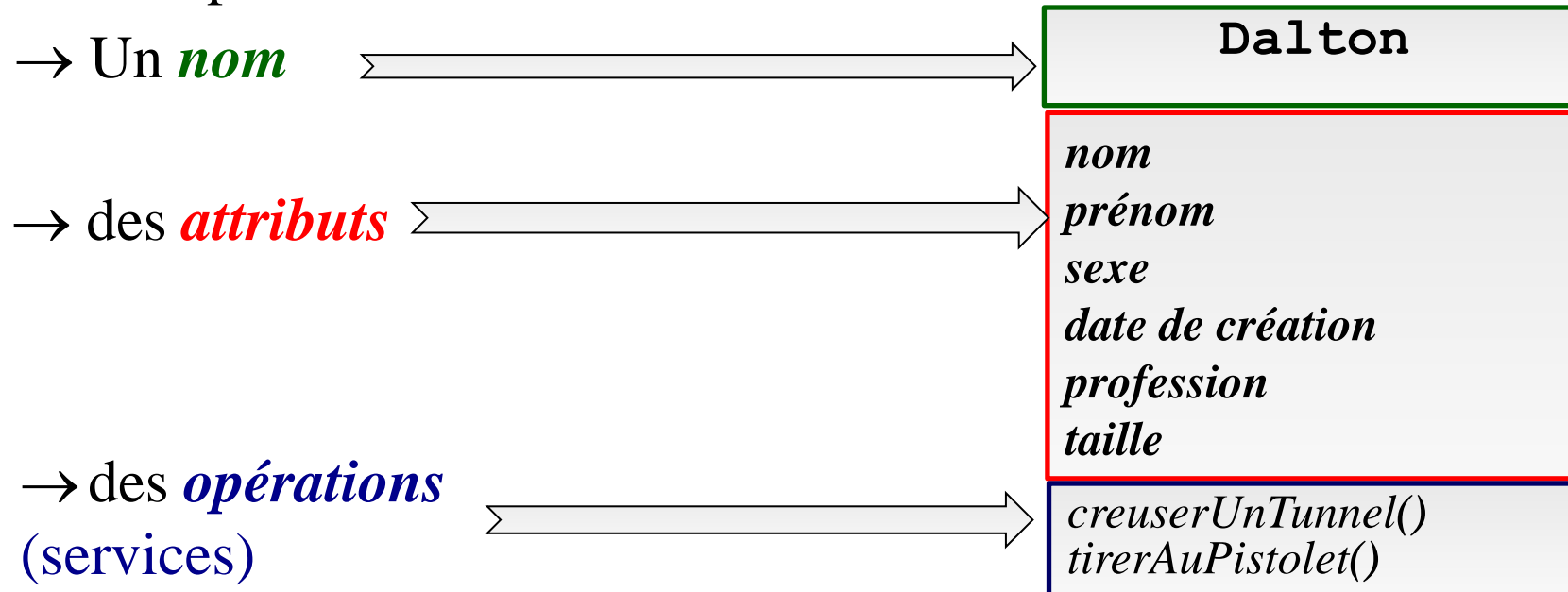
*creuserUnTunnel()*  
*tirerAuPistolet()*



# Introduction à la notion de classe via la classe Dalton

Une classe permet donc de **regrouper des objets** qui présentent des propriétés similaires (*attributs*) et des comportements communs (*opérations*)

Une Classe possède :



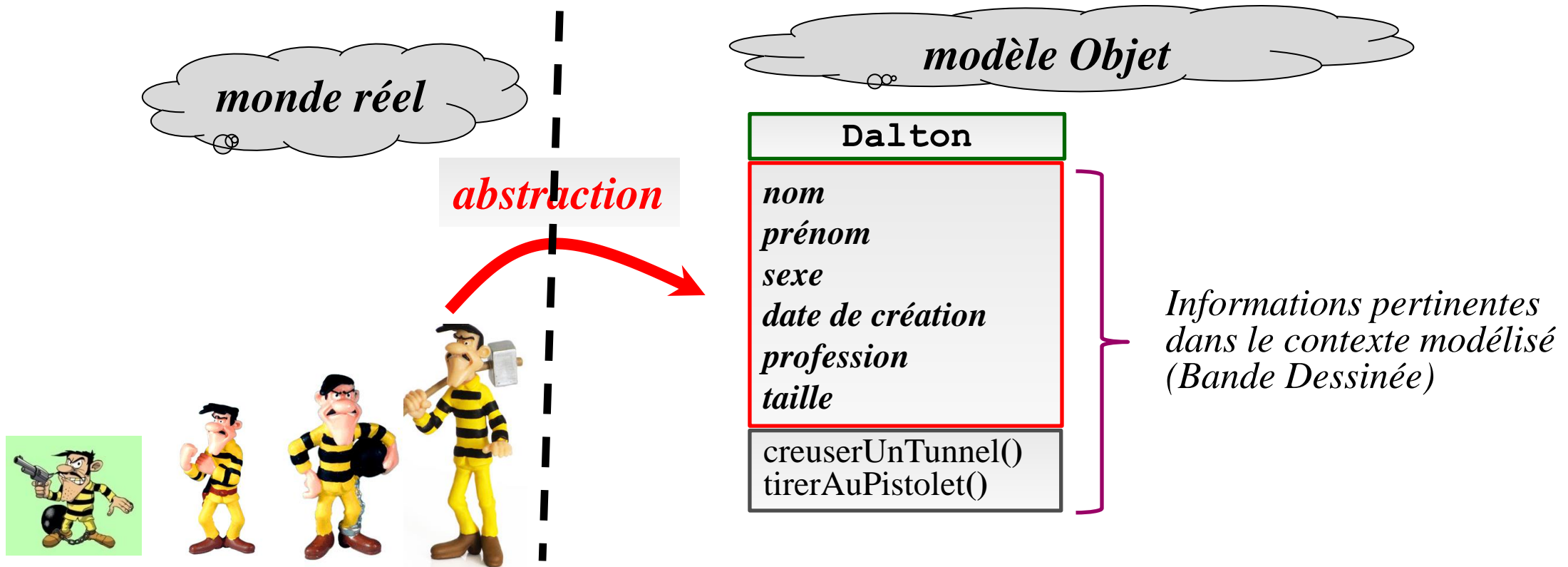
La classe est une sorte de **moule** à partir duquel sont créés des objets  
⇒ L'objet «sera **personnalisé**» par les **valeurs** données aux **propriétés** de la classe



# Une classe est une **abstraction** ...

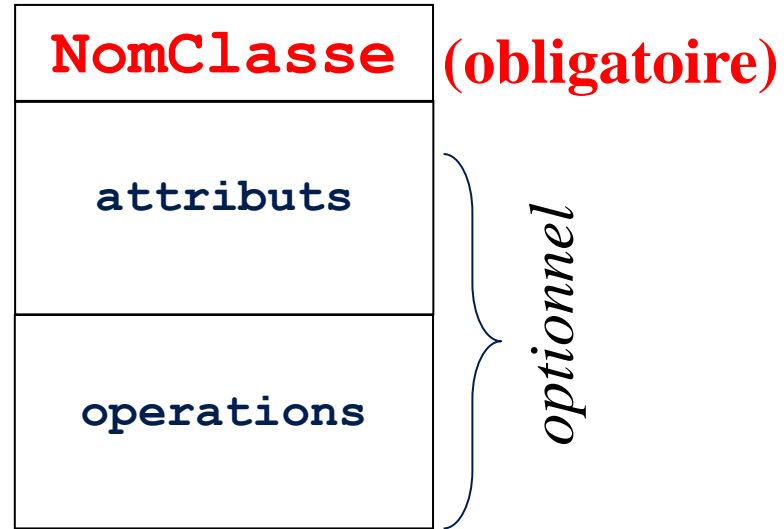
**Abstraction:** *Opération intellectuelle qui consiste à isoler par la pensée l'un des caractères de quelque chose et à le considérer indépendamment des autres caractères de l'objet.*

⇒ Une **classe** est une **abstraction** qui décrit l'état et le comportement des futurs objets de la classe .



⇒ **Abstraction = 1 point de vue de la réalité**

# Formalisme UML pour une classe



Le nom des classes *abstraites* devra être écrit en *italique*.

Syntaxe pour les **attributs** :

```
[visibilité] nom [: type] [multiplicité] = [valeurInitiale] [{propriété}]
```

Syntaxe pour les **opérations** :

```
[visibilité] nom [(listeParametres)] [: typeRetour] [{propriété}]
```

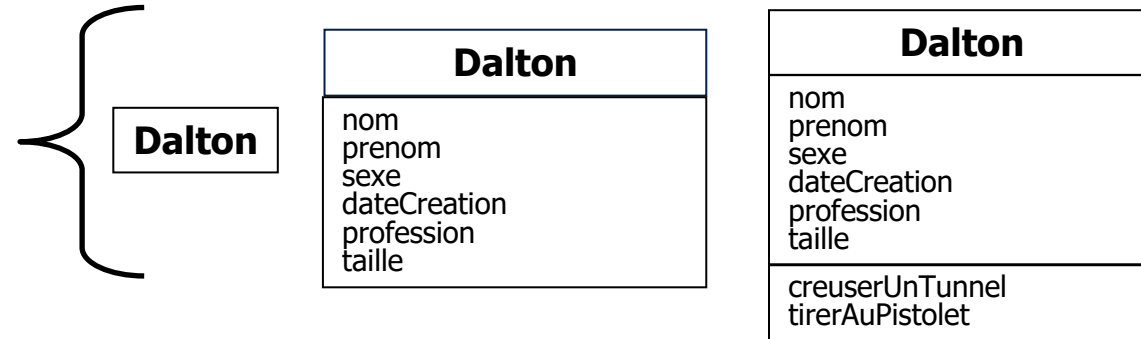
*Par convention :*

- le *nom de la classe* commencera par une *majuscule*.
- le *nom des attributs* et des *opérations* commencera par une *minuscule*

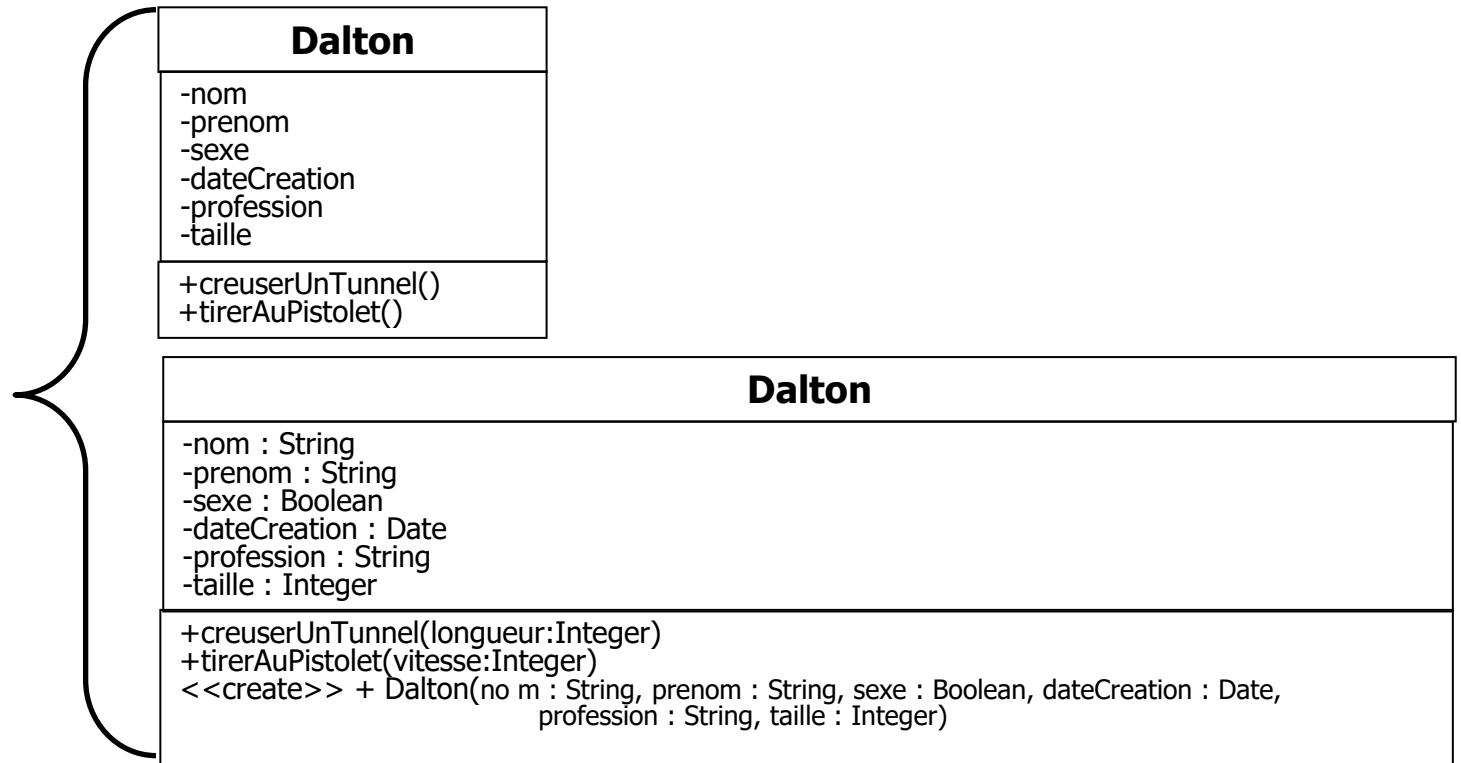
# Granularité dans la modélisation des classes ...

*... Plus ou moins de détails suivant les besoins de la modélisation...*

Modélisation « gros grain »  
plutôt utilisées  
**en phase d'Analyse**



Modélisation à « granularité plus fine »  
plutôt utilisées  
**en phase de Conception**



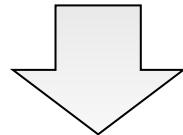
# Principe d'encapsulation

## Principe d'encapsulation

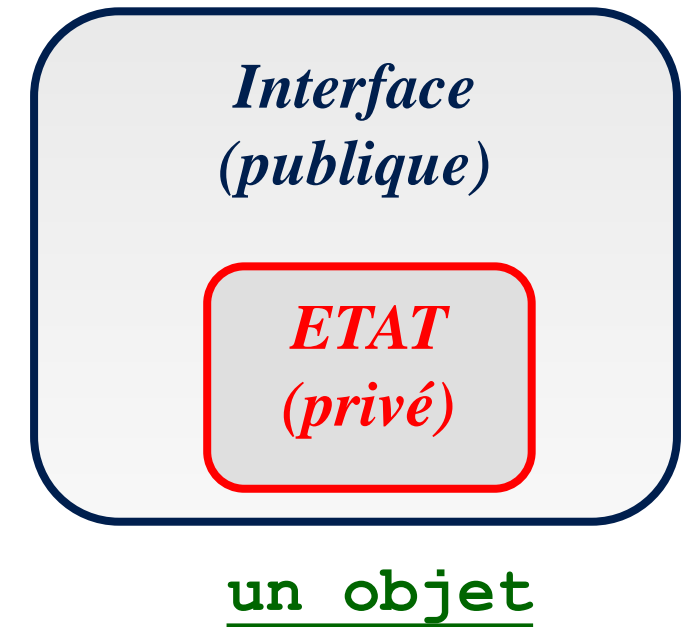
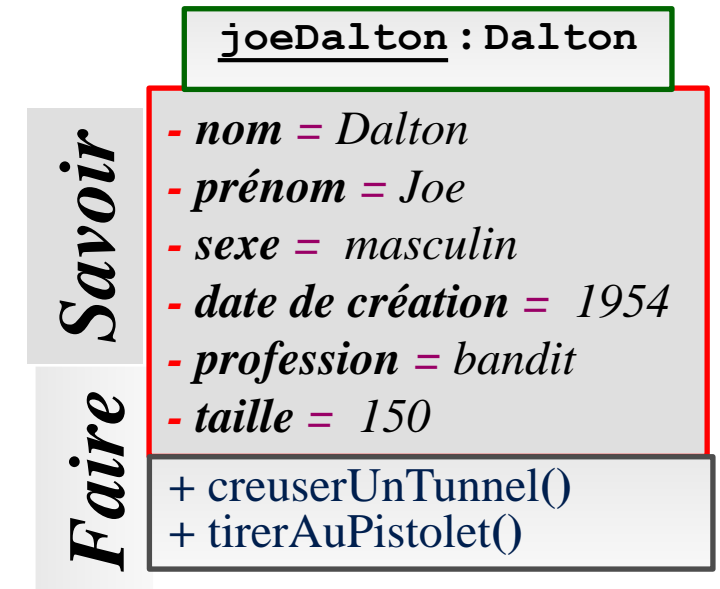
L'encapsulation est un mécanisme consistant  
**à réunir l'intérieur d'une même entité :**  
**les données et les opérations.**

⇒ seuls les services, qui définissent l'**interface** de l'objet,  
sont accessibles de l'extérieur (opérations **publiques** (+) )

⇒ l'état est caché, c.à.d. non modifiable directement  
(attributs de type **privés** (-) )



*Notion de Visibilité*



# Notion de Visibilité

La **visibilité** permet dans une classe de spécifier ce qu'elle laisse voir de ses membres (attributs et opérations) et dans quelles conditions

**Plusieurs degrés d'encapsulation** peuvent être définis pour les membres d'une classe (**niveaux d'accès**) :



Public

Protected

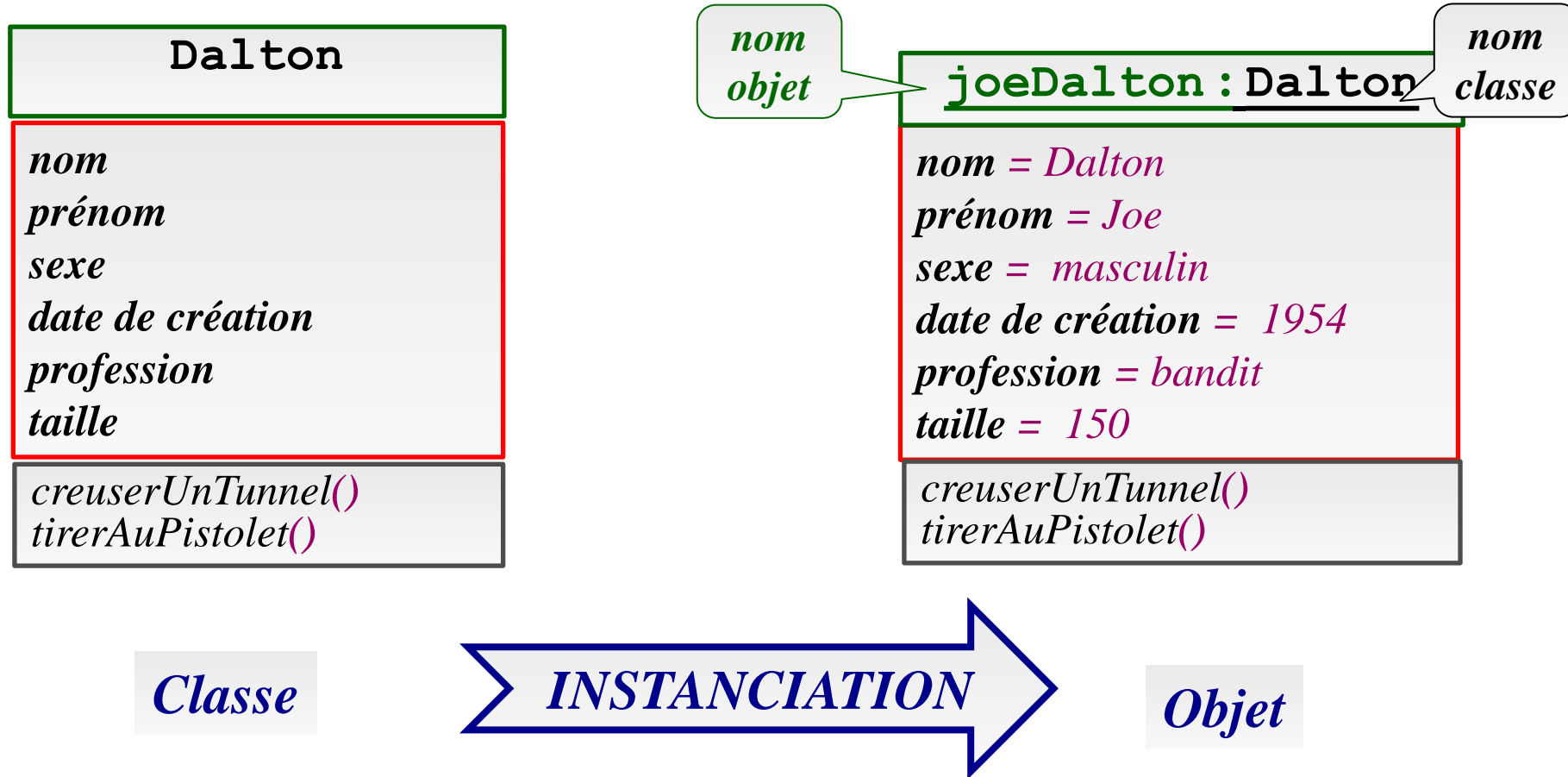
Private

⇒ **+ public :** *membre visible (accessible) par toutes les classes sans aucune restriction*

⇒ **# protected :** *membre visible dans la classe où il est déclaré et dans toutes ses classes filles (notion d'héritage)*

⇒ **- private :** *membre seulement visible à l'intérieur de la classe où il est déclaré.*

# A propos de l'Instanciation

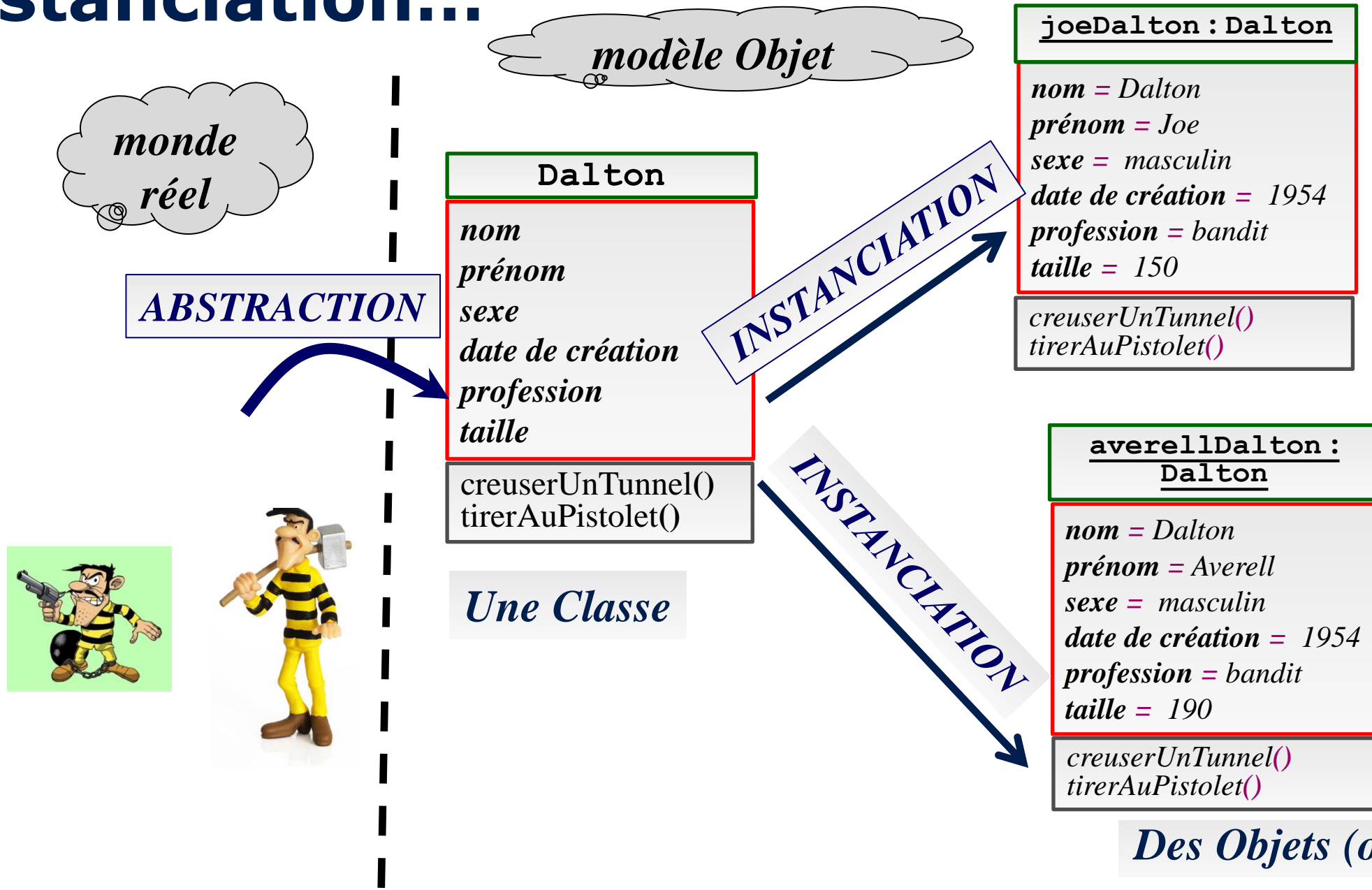


**L'instanciation : action de créer un objet à partir d'une classe**

Un **objet** est aussi appelé une « **instance de classe** »



# Récapitulatif des notions d'abstraction & d'instanciation...



# **Les Objets**

## **dans la modélisation UML**

# Vie des objets

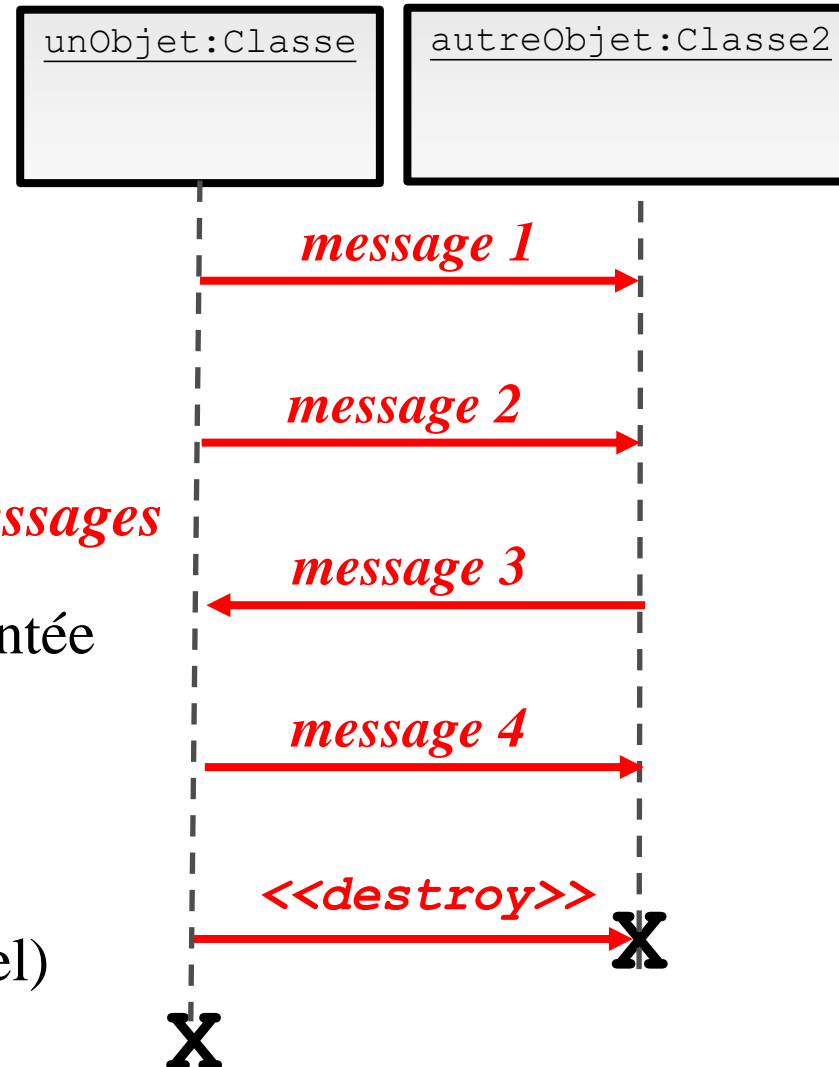
Les *objets* sont les agents *effectifs* et « *vivants* » du programme :  
*ils naissent, ils interagissent, ils meurent*

*Naissance* des objets  
via le mécanisme d'*instanciation*  
(stéréotype <<create>> optionnel)

*Vie* : les objets interagissent  
en s'envoyant des *messages*

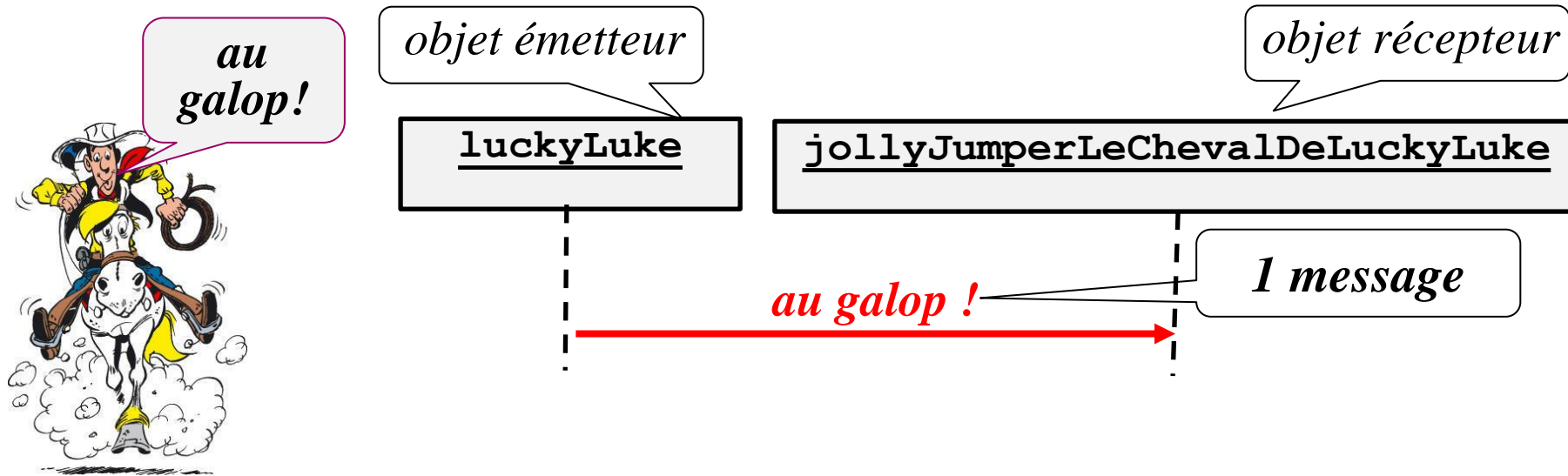
⇒ *Une ligne de vie* peut être représentée

*Mort* des objets  
(le stéréotype <<destroy>> optionnel)



# Message entre objets

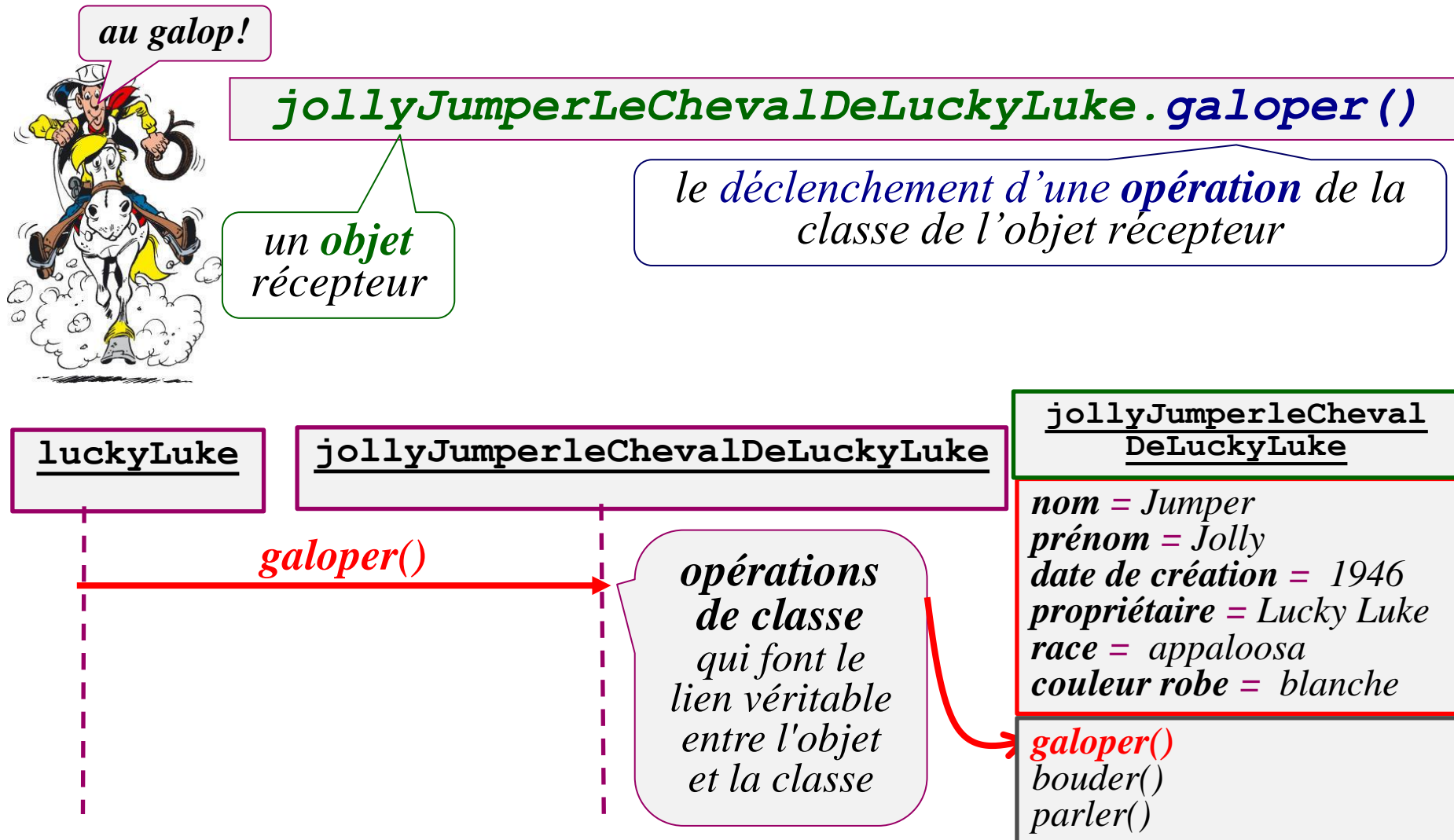
*Les objets interagissent et communiquent entre eux  
par l'envoi de messages*



Durant sa vie, l'objet va devoir réagir aux **messages** qu'on lui envoie.  
La **façon** dont le message est traité détermine le **comportement de l'objet**.

# Du *message* entre Objets à l'*opération* de Classe

Lancer un *message synchrone* à un objet revient à provoquer le *déclenchement d'une opération* définie *dans la classe* de cet objet.



# Un peu plus sur les *messages* et *opérations*

Un message peut être envoyé en **mode synchrone** ou **asynchrone**

***Emetteur bloqué en attente de réponse***  
(Ex : Appel opération)  
Le plus courant en UML

***Pas d'attente de réponse*** (Signal)

Un objet ne réagit pas toujours de la même façon **à un même message**  
Sa réaction dépend de l'état dans lequel il se trouve.

Dans les langages de Programmation Orientée Objet,  
l'implémentation de l'opération donnera lieu à une **méthode**

*Sémantiquement : opération (COO) ≠ méthode (POO)*



# Récapitulatif des messages aux opérations de classe



*au pas !*



*au trot !*



*au galop !*

luckyLuke

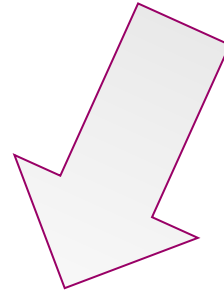
jollyJumperLeChevalDeLuckyLuke

*au pas !*

*au trot !*

*au galop !*

*3 messages*



luckyLuke

jollyJumperleChevalDeLuckyLuke

*marcher()*

*trotter()*

*galoper()*

*3 opérations  
de classe*

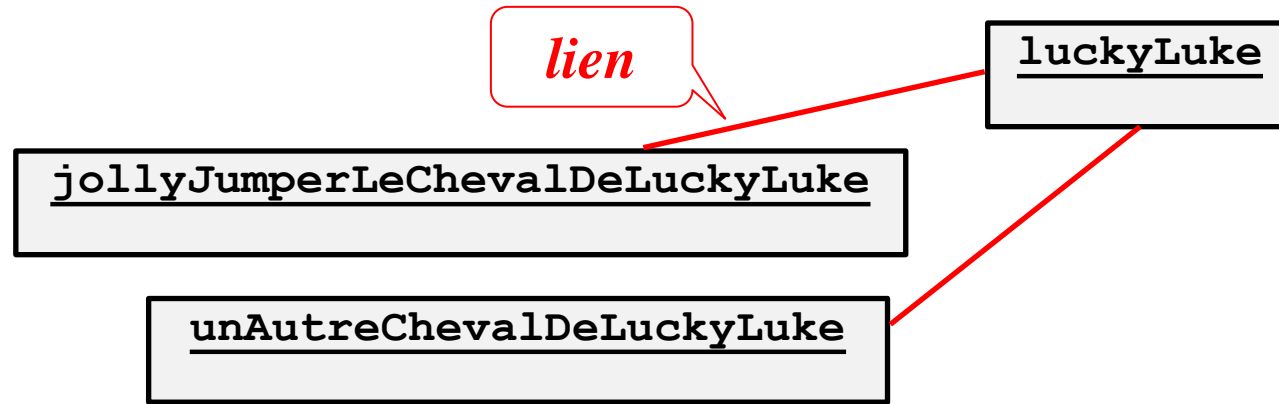
leChevalDeLuckyLuke

*nom = Jumper  
prénom = Jolly  
date de création = 1946  
propriétaire = Lucky Luke  
race = appaloosa  
couleur robe = blanche*

*marcher()  
trotter  
galoper()  
bouder()  
parler()*

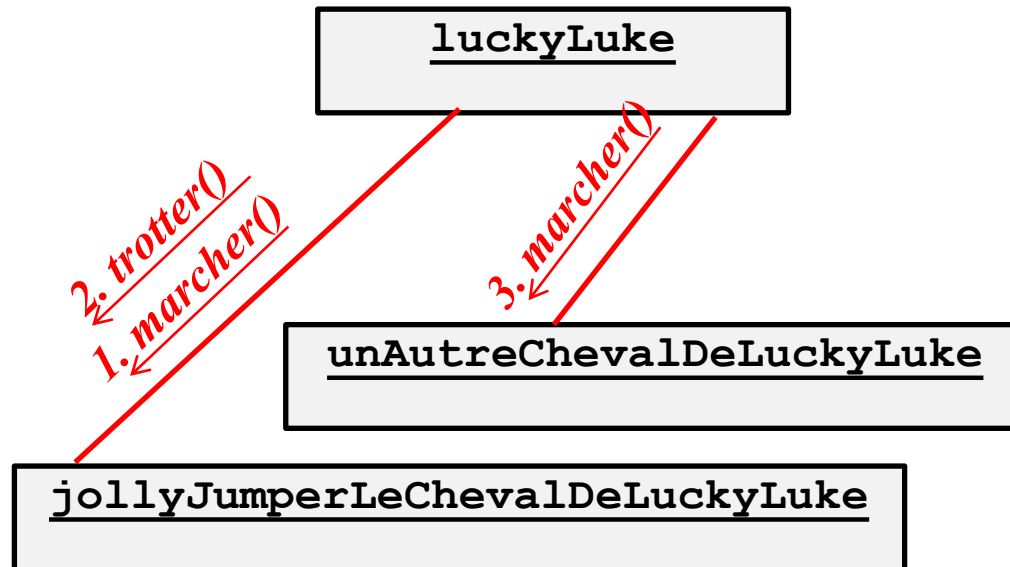
# Liens entre Objets

→ Les objets sont *connectés par « des liens »*.



Cette *modélisation statique* correspond en UML diagramme d'objets.

→ Un lien est un vecteur pouvant supporter l'envoi de message entre les objets.



Cette *modélisation dynamique* correspond en UML au diagramme de communication

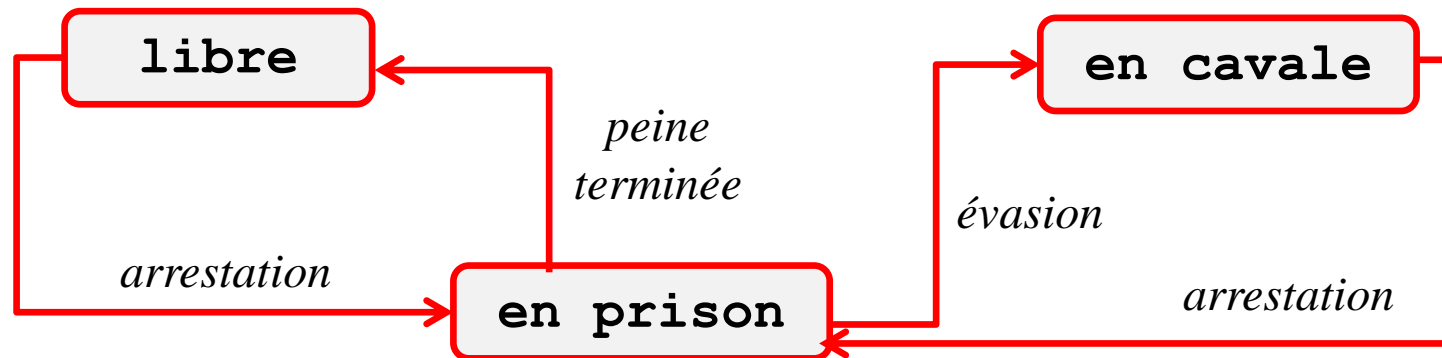
# Etat d'un Objet

Un objet passe par une succession d'*états* durant son *Cycle de Vie*.

Exemple : Trois états peuvent être identifiés pour un objet de la classe **Dalton** :  
**libre**, **en prison** et **en cavale**



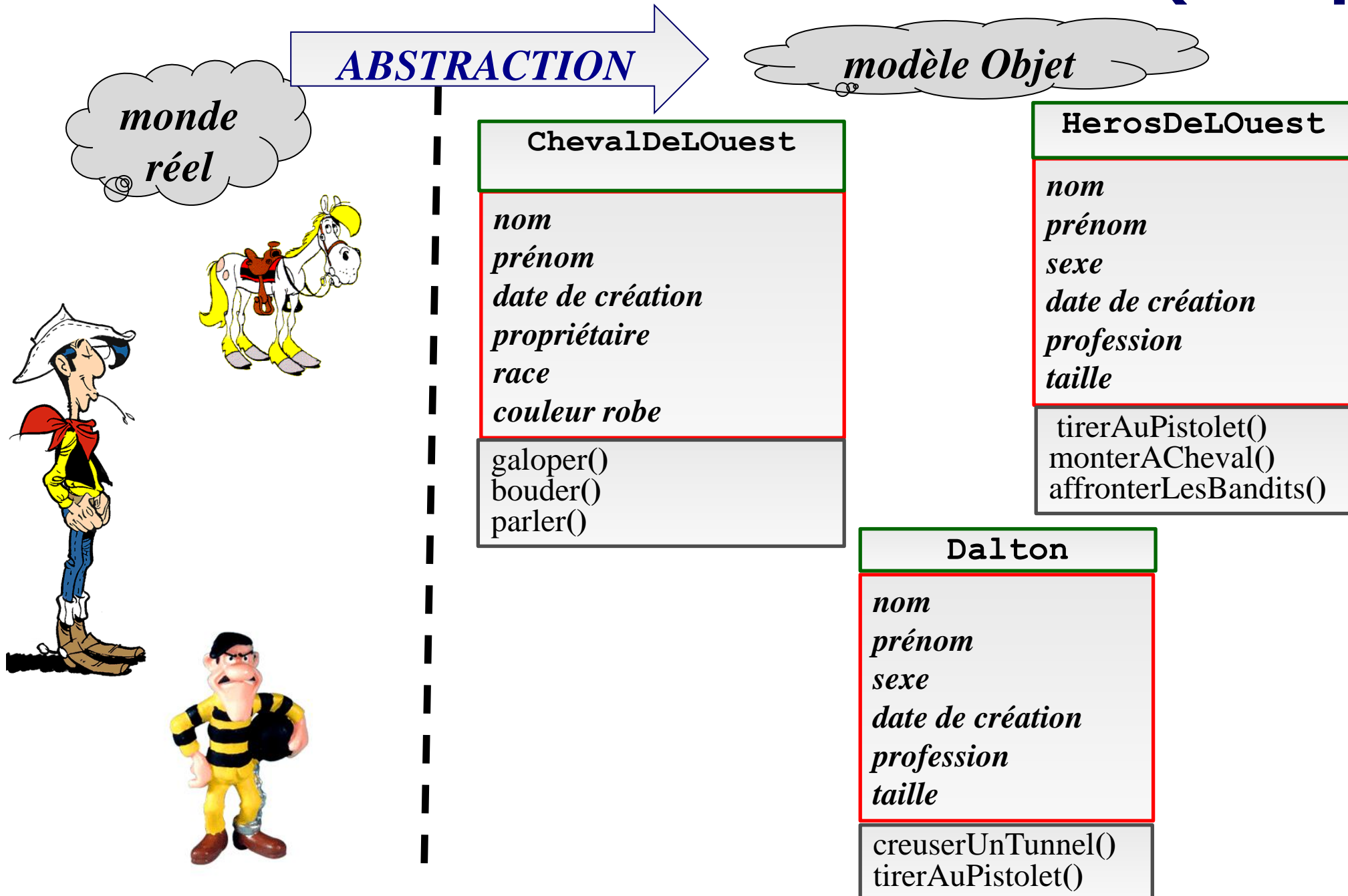
Pour décrire le *Cycle de Vie commun aux Objets d'une même classe*, UML propose un diagramme d'états –transitions.



Un état a une *durée finie*,  
Variable *en fonction des événement* qui lui arrivent.

# **Introduction au diagramme de classes**

# Abstraction des classes de ce cours (1<sup>ère</sup> passe)

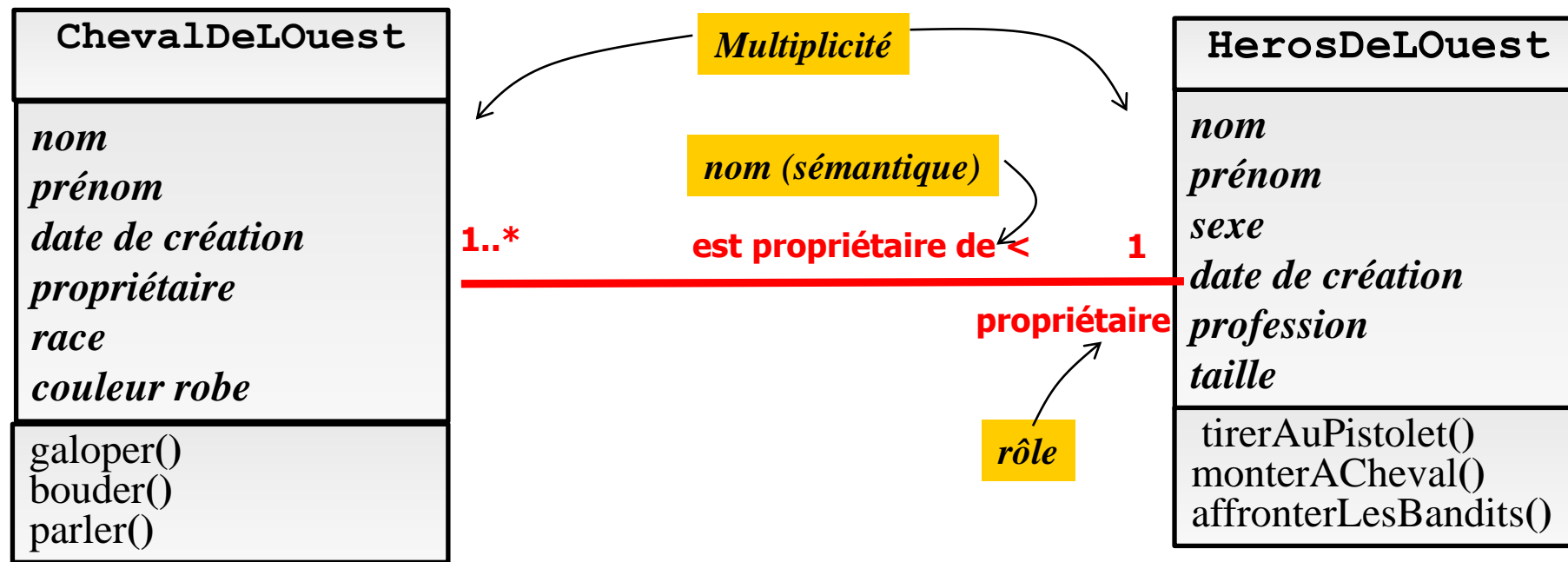


# Associations entre Classes

Une association lie deux classes ( $\neq$  un lien lie deux objets )

⇒ Une association décrit un ensemble de liens.

*L'extrémité d'une association peut porter une indication de **multiplicité***



Une association peut posséder un **nom** (avec un sens de lecture < ou >)

ou

L'extrémité d'une association peut posséder un nom de **rôle**, rôle joué par la classe dans l'association



# Diagramme de classes vs diagramme d'objets

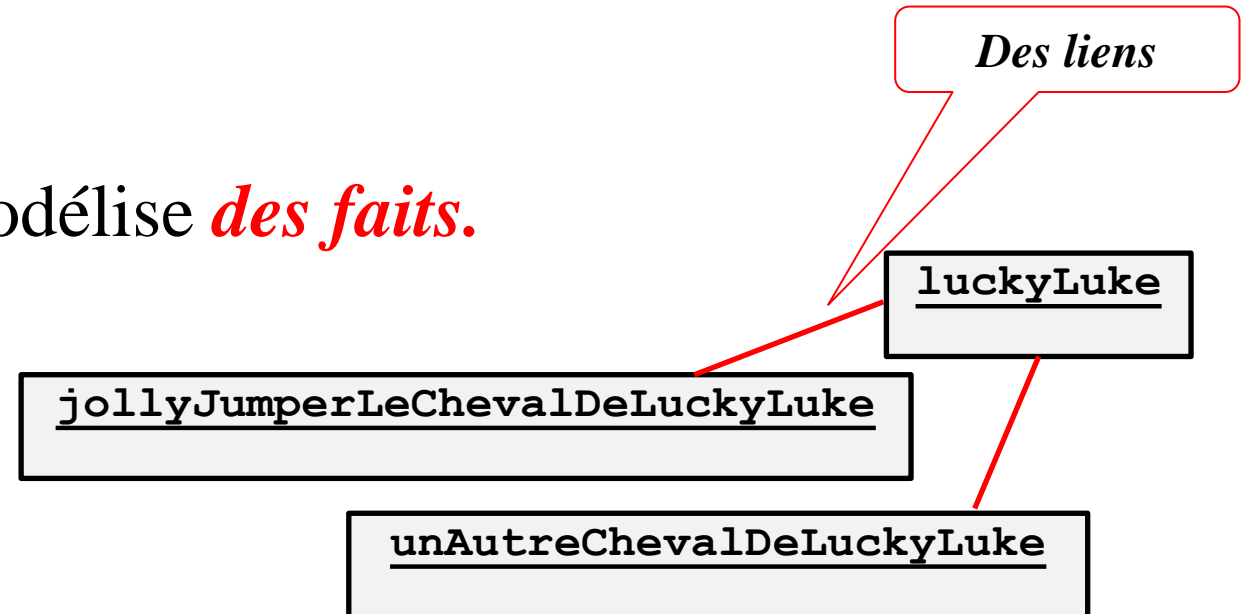
*Le diagramme de classes* modélise **les règles**



*1 association*

*Le diagramme d'objets* modélise **des faits.**

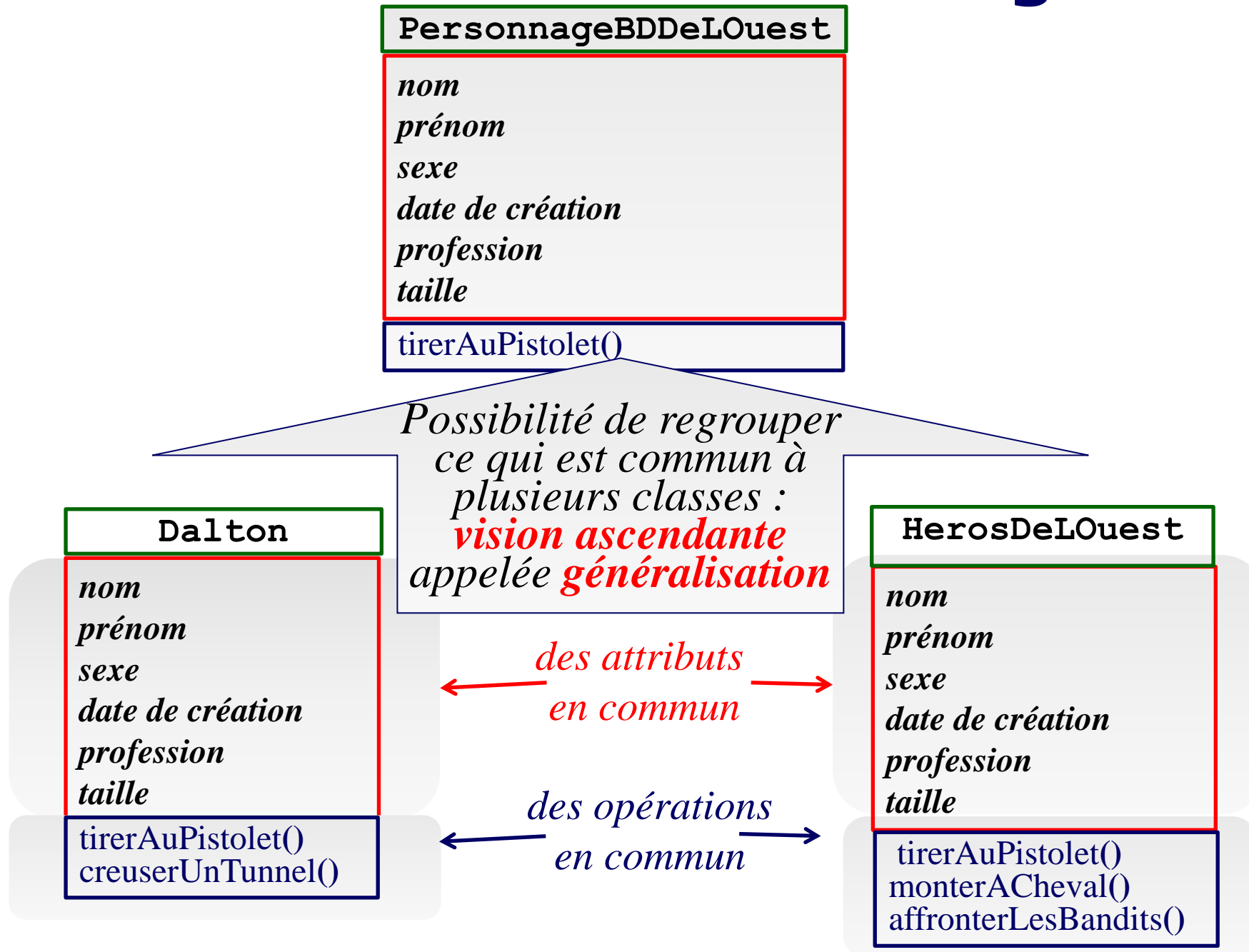
→ Il décrit, *à un instant  $t$ , un état du système* en montrant les instances créées, leur état et les liens entre elles.



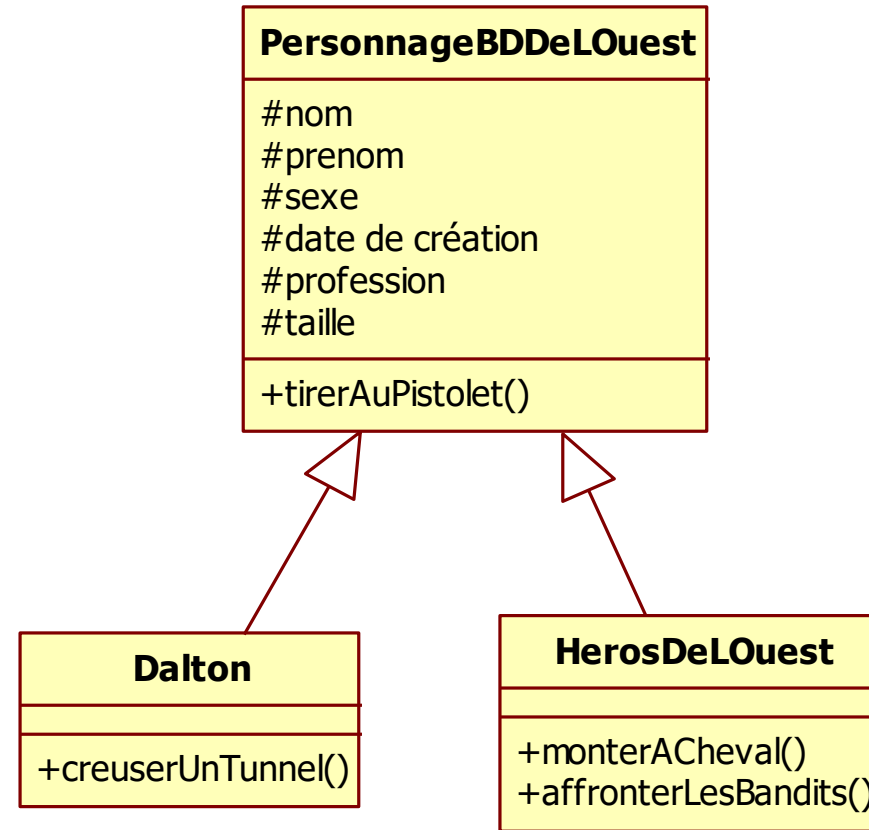
*Des liens*


→ Il doit toujours rester conforme au modèle de classes

# Introduction à la notion d'Héritage



# Héritage : Notation UML

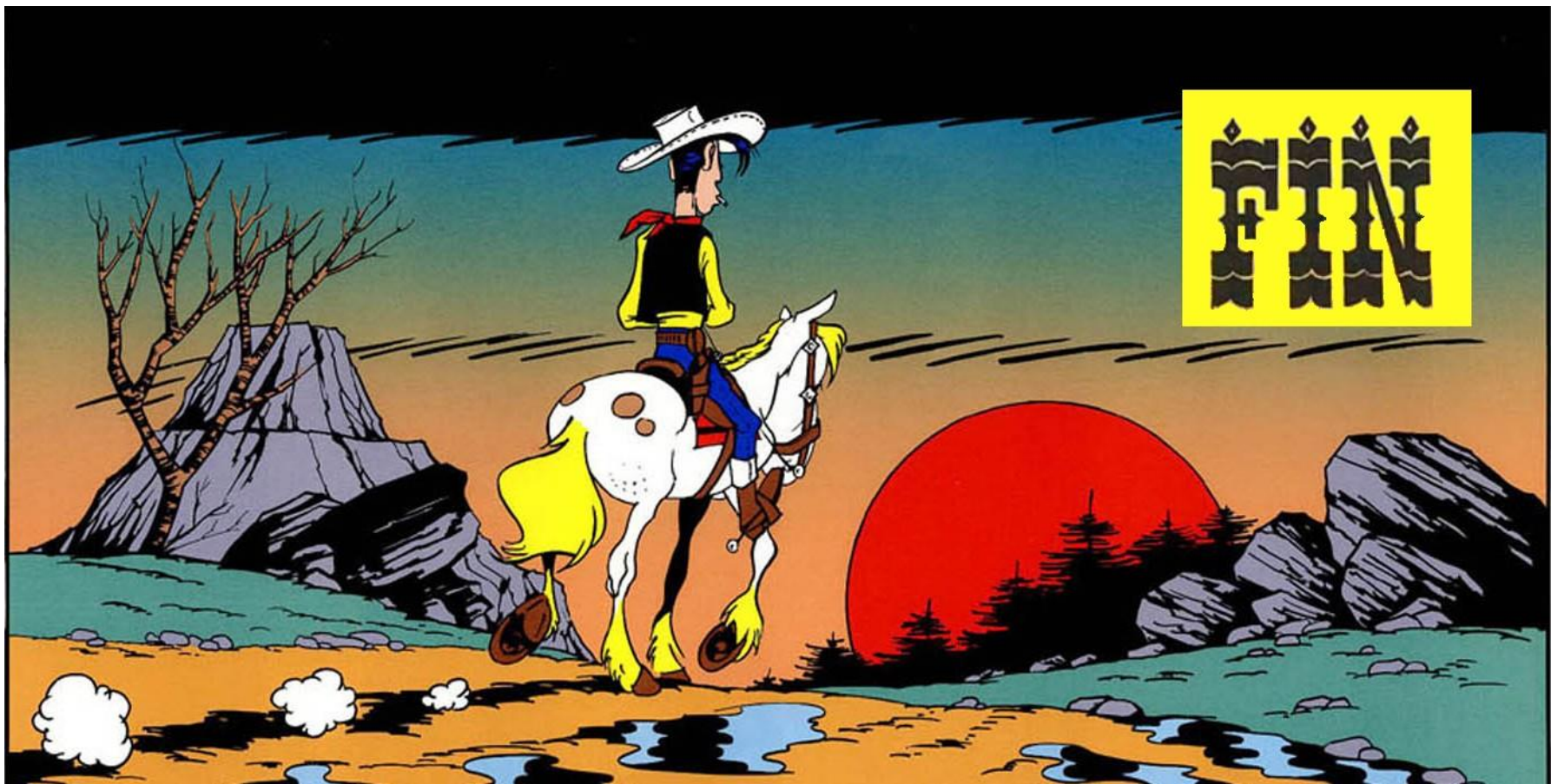


La notation UML pour **l'héritage de classe** est : 

⇒ C'est une relation de **généralisation** qui exprime une relation **EST-UN** (extends)

Un **Dalton** **EST-UN** **PersonnageBDDeLOuest**

Un **HerosDeLOuest** **EST-UN** **PersonnageBDDeLOuest**



# Les principes de Bases de l'Orienté Objet

**Classe** : **Abstraction des caractéristiques communes à un ensemble d'objets.**

Une classe permet de créer des **objets** qui communiquent entre eux par des **messages**.



**Encapsulation** : **Protection de l'information contenue dans un objet.**

Principe permettant de **regrouper** les *données* et les *routines* permettant de lire ou manipuler ces données.

En POO, le but de l'encapsulation est de **masquer** les **attributs** et certaines **méthodes** afin de ne rendre disponible que le comportement souhaité.

**Héritage** : **Transmission des caractéristiques à ses descendants.**

Mécanisme permettant, lors de la déclaration d'une nouvelle classe (fille), d'y inclure les caractéristiques d'une autre classe (mère).

**Polymorphisme** : du grec « **qui peut prendre plusieurs formes** »

Concept consistant à fournir une interface unique à des entités pouvant avoir différents **types**.

→ **polymorphisme d'héritage** (redéfinition, spécialisation ou *overriding*)

→ **polymorphisme ad hoc** (surcharge ou *overloading*)

→ **polymorphisme paramétrique** (généricité ou *template*)

# **Annexe**





# Une petite vidéo ...



<https://www.youtube.com/watch?v=50VrRVp7CtY>

*Images extraites :*

<http://goutdumonde.blogspot.fr/2008/11/tajine-dossier-complet.html>

<http://muffintop.over-blog.com/article-tajine-d-agneau-aux-pruneaux-et-aux-olives-98207598.html>

Isabelle BLASQUEZ