

# R2.01 - Développement Orienté Objets

## Diagrammes de séquence UML

Arnaud Lanoix Brauer  
Arnaud.Lanoix@univ-nantes.fr



Département informatique

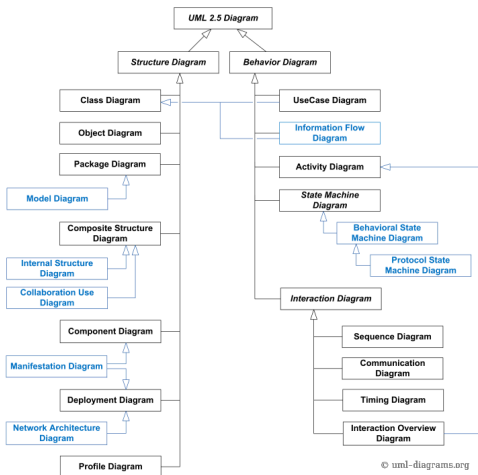
# Diagrammes en UML 2.5

## Diagrammes de structure (statiques)

- **Diagramme de classes**
- **Diagramme d'objets**
- Diagramme de paquets
- Diagramme de structure composite
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de profils

## Diagrammes de comportement (dynamiques)

- Diagramme des cas d'utilisation
- Diagramme d'activité
- Diagramme états-transitions
- *Diagrammes d'interaction*
  - ▶ **Diagramme de séquence**
  - ▶ Diagramme de communication
  - ▶ Diagramme de temps
  - ▶ Diagramme global d'interaction



© uml-diagrams.org

- 1 Diagramme d'objets
- 2 Diagramme de séquence

# Diagramme d'objets

Un **diagramme d'objets** UML représente des **instances** de classes, c-à-d des objets :

- est **forcément** associé à un **diagramme de classes**
  - ▶ **Attention** : les différents diagrammes doivent être **cohérents**
- illustre l'**état** d'un l'objet (à un moment donné)
- illustre les **associations** entre les différents objets (à un moment donné)

= **photographie (instantanée)** des objets instanciés

- permet d'illustrer une configuration particulière
- utile lorsque les associations en jeu entre objets sont complexes

# Diagramme d'objets

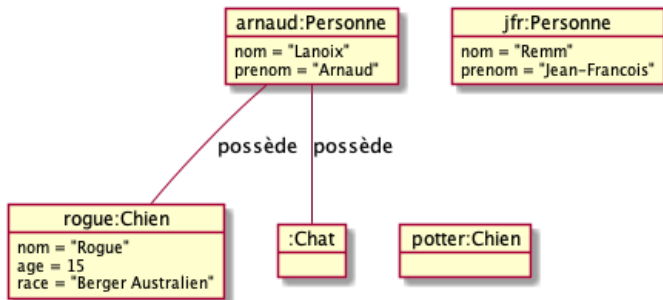
Un **diagramme d'objets** UML représente des **instances** de classes, c-à-d des objets :

- est **forcément** associé à un **diagramme de classes**
  - ▶ **Attention** : les différents diagrammes doivent être **cohérents**
- illustre l'**état** d'un l'objet (à un moment donné)
- illustre les **associations** entre les différents objets (à un moment donné)

= **photographie (instantanée) des objets instanciés**

- permet d'illustrer une configuration **particulière**
- utile lorsque les associations en jeu entre objets sont **complexes**

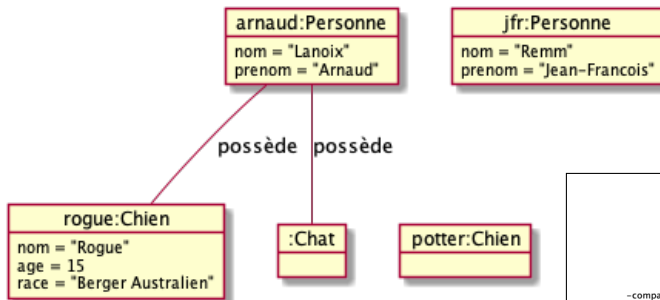
# Exemple de diagramme d'objets



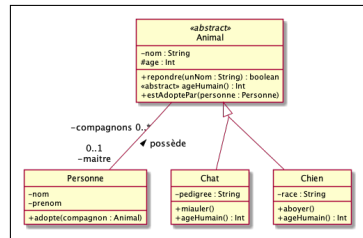
- on peut **identifier** (ou pas) les objets :  
rogue:Chien
- on peut **valuer** (ou pas) les attributs d'un objet

Les deux diagrammes sont **cohérents**

# Exemple de diagramme d'objets



- on peut **identifier** (ou pas) les objets :  
rogue:Chien
- on peut **valuer** (ou pas) les attributs d'un objet



Les deux diagrammes sont **cohérents**

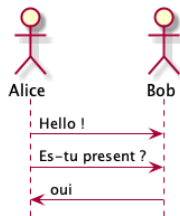
- 1 Diagramme d'objets
- 2 Diagramme de séquence



# Diagramme de séquence

Un **diagramme de séquence** UML illustre les **interactions** entre **acteurs** de manière chronologique.

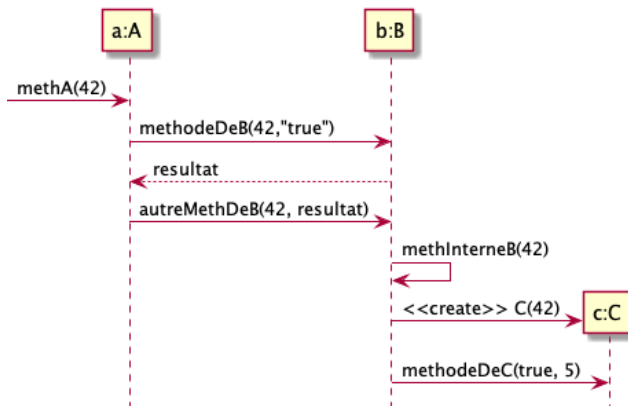
- Le "passage du temps" est représenté **verticalement**



Nous concernant

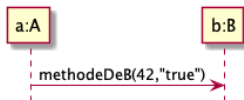
- Les acteurs seront des **objets** (instances de classes) qui interagiront
- Interaction = appel de méthodes
- sera **forcément** associé à un **diagramme de classe**
  - Attention** : les différents diagrammes devront être **cohérents**

# Syntaxe des diagrammes de séquence



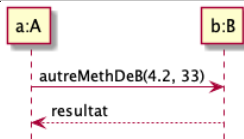
- Les **acteurs** (en haut) sont des instances de classes
- Le **temps** est représenté par les lignes pointillées verticales
- Les appels de méthodes s'enchaînent **successivement**

# Details



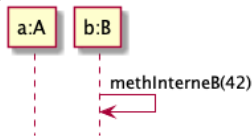
Appel de méthode :

a instance de A appelle la méthode `methodeDeB(...)` sur l'instance b de la classe B



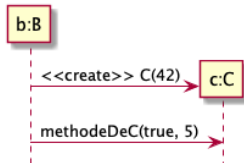
Appel avec retour :

l'appel à la méthode `autreMethDeB(...)` renvoie un resultat à l'instance appelante



Appel à une méthode "de soi-même" :

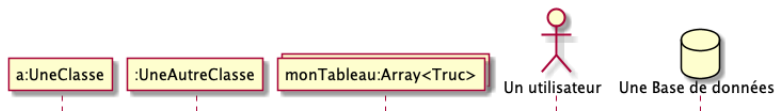
b:B appelle un autre méthode de la classe B



Instantiation d'un nouvel objet :

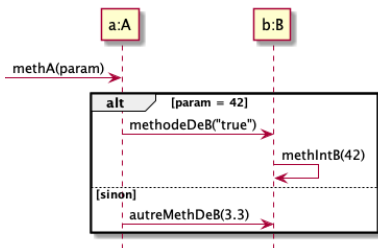
b:B crée une nouvelle instance de C en appelant son constructeur

# Différents acteurs dans un diagramme de séquence



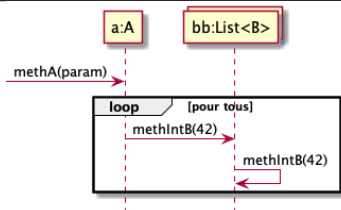
- Les acteurs peuvent être des **instances de classes**, **nommées** ou non
- Un acteur peut aussi représenter une **collection** d'objets d'un type donné
- Un acteur peut être un **utilisateur** du système modélisé
- Un acteur peut être une **base de données**
- ...

# Fragments combinés



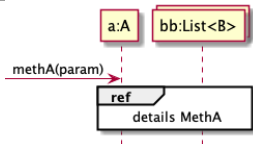
alt pour alternative : exprime un comportement **conditionnel**

- Les conditions [...] sont exprimées en langue naturelle



loop : exprime une **boucle**

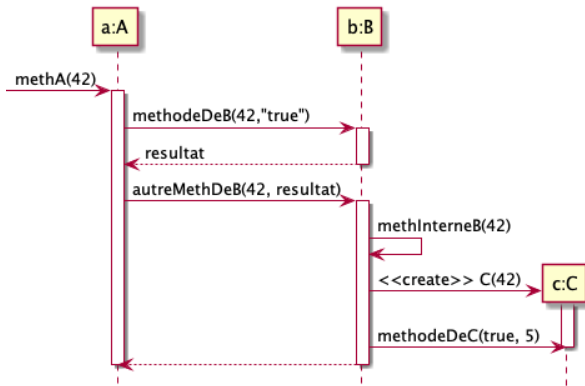
- La condition [...] est exprimée en langue naturelle



ref : permet de **référencer** un autre diagramme

# Lignes de vie

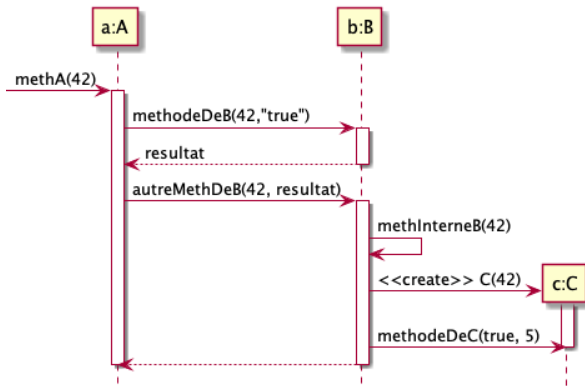
= "rectangles superposés sur la ligne de temps"



- permet d'indiquer les périodes d'activité des différents acteurs
- pas très utile dans notre contexte

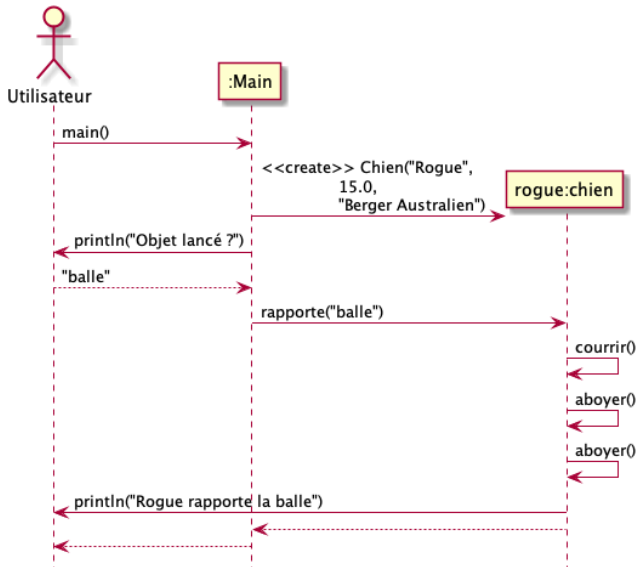
# Lignes de vie

= "rectangles superposés sur la ligne de temps"



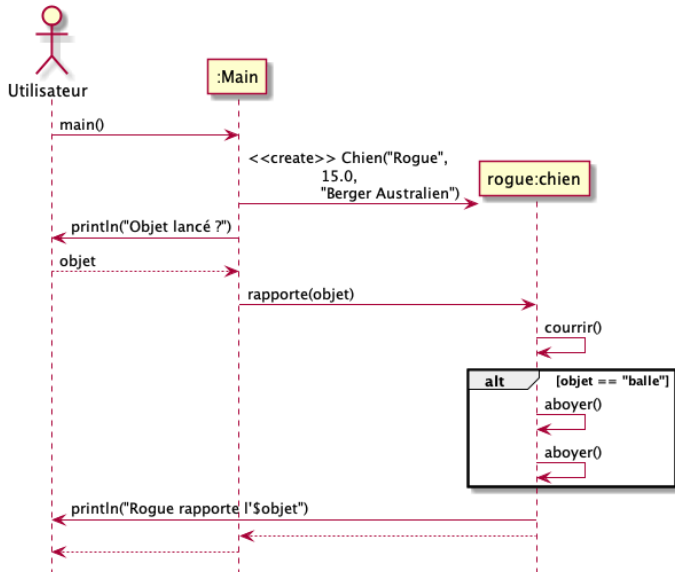
- permet d'indiquer les périodes d'activité des différents acteurs
- pas très utile dans notre contexte

# Exemple

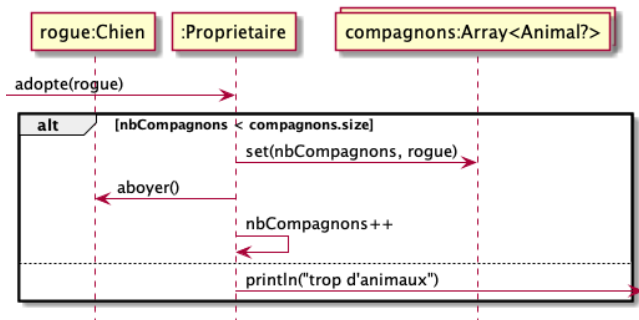




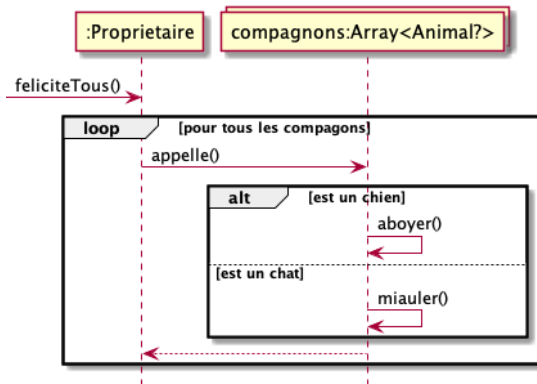
# Exemple : alt



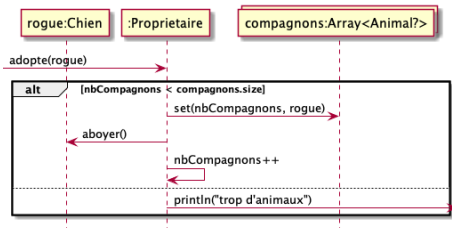
## Exemple : alt (2)



# Exemple : loop



# Diagramme de séquence + code Kotlin

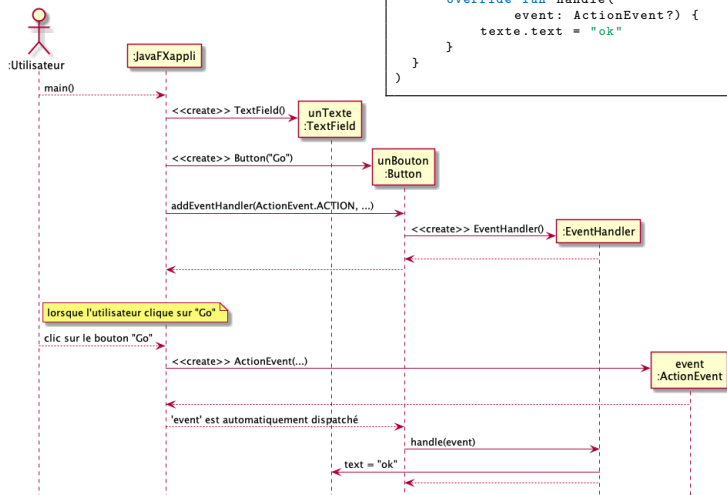


```
class Proprio (...) {
    compagnons : Array<Animal?>
    nbCompagnons : Int = 0
    ...
    fun adopte(nouveau : Chien) {
        if (nbCompagnons < compagnons.size) {
            compagnons.set(nbCompagnons, nouveau)
            nouveau.aboyer()
            nbCompagnons++
        }
        else {
            println("trop d'animaux")
        }
    }
}
```

# Diagramme de séquence + code Kotlin (exemple JavaFX)

```
val unTexte = TextField()
val unBouton = Button("Go")
```

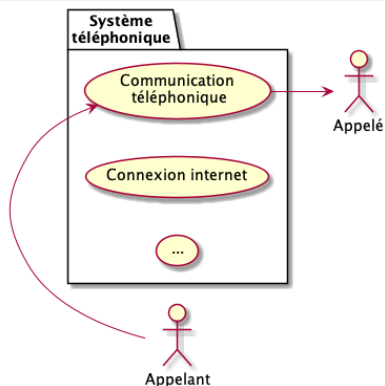
```
unBouton.addEventHandler(
    ActionEvent.ACTION,
    object : EventHandler<ActionEvent> {
        override fun handle(
            event: ActionEvent?) {
                texte.text = "ok"
            }
        }
    )
)
```



# Un autre usage : les cas d'utilisation

## Cas d'utilisation + sequences

Les diagrammes de séquence peuvent AUSSI servir à **illustrer/documenter** des **cas d'utilisation**

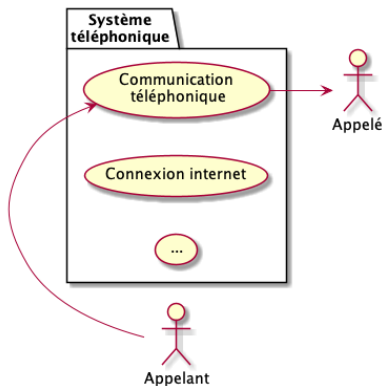


cas d'utilisation : Com. téléphonique  
Scénario nominal :

# Un autre usage : les cas d'utilisation

## Cas d'utilisation + sequences

Les diagrammes de séquence peuvent AUSSI servir à **illustrer/documenter** des **cas d'utilisation**



**cas d'utilisation : Com. téléphonique**  
**Scénario nominal :**

