CLOO

01.06.2004

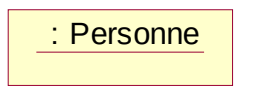
Modèles dynamiques

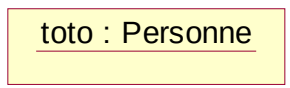
- UML propose 4 diagrammes pour représenter la dynamique du système. Ce sont les diagrammes:
  - D'intéraction: qui collabore avec qui ?
    - Diagramme de séquence
    - Diagramme de collaboration
  - D'états-transitions:
    - « réactions » d'un objet lors de la réception de messages
  - D'activités:
    - décomposition des traitements en activités et distribution sur les objets

Sequence diagram

- Représentation temporelle de l'échange de messages pour la réalisation d'une tâche

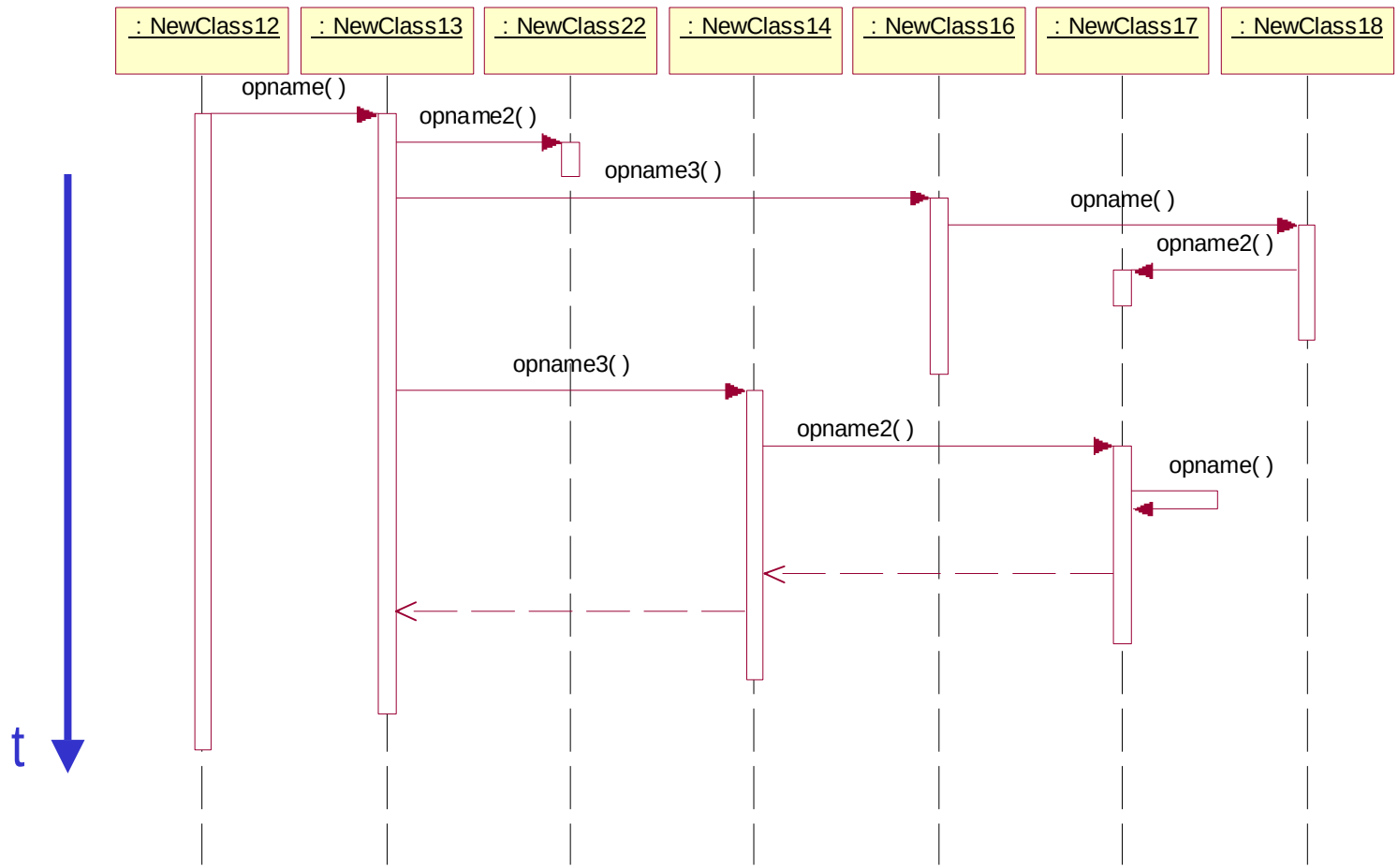
- 2 dimensions
  - Horizontalement: instances
    - Rectangle avec le nom de la classe souligné
    - Indication possible du « nom » de l'instance





  - Verticalement: temps

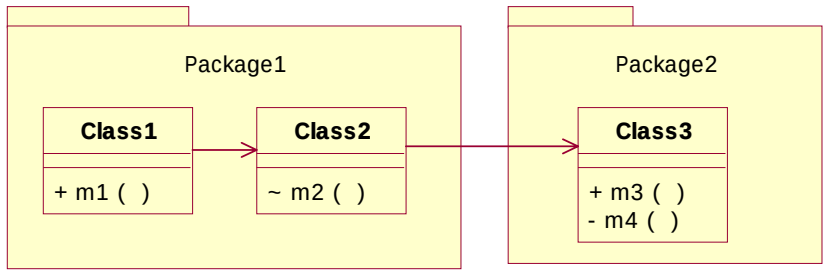
Sequence diagram

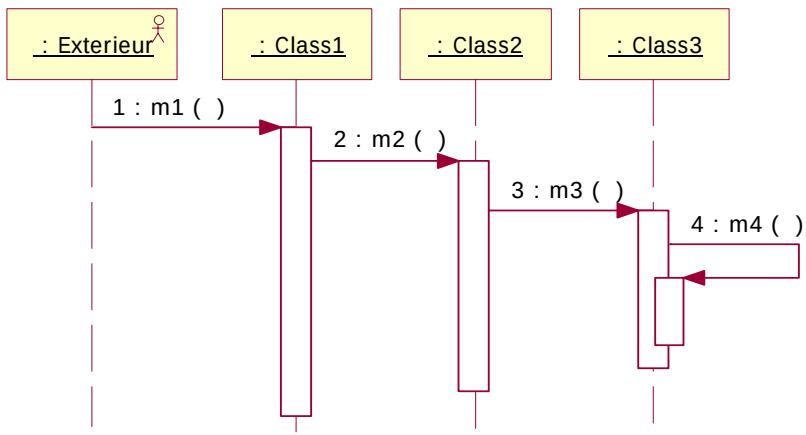


Remarques I

- Les instances ne peuvent communiquer que si elles se « connaissent »
  - Il existe une association entre leurs classes (collaboration)
  - Un objet est retourné comme value de l'invocation d'un message
- Les messages envoyés doivent faire partie du protocole visible par l'objet qui invoque la méthode.
- Un objet peut s'envoyer un message
- Il est possible de représenter la valeur retournée par un message.

SD et visibilité

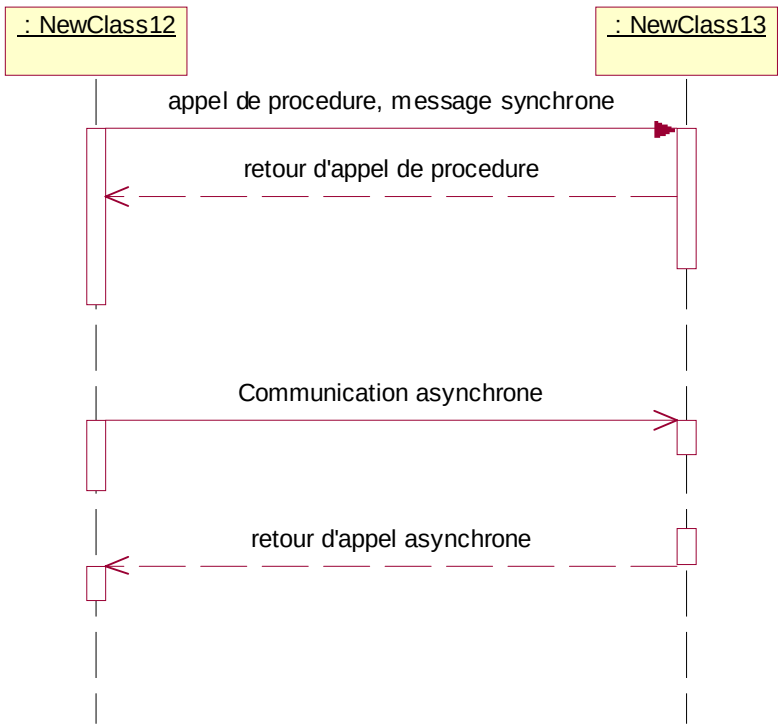




Remarques II

- La durée d'exécution (*focus of control*) est indiquée par un rectangle vertical sur la ligne de l'instance.
- La distance verticale entre deux messages n'a pas de signification
- L'ordonnancement horizontal des instances n'a pas de signification.
- C'est le séquence diagram qui permet de valider la directionalité des associations

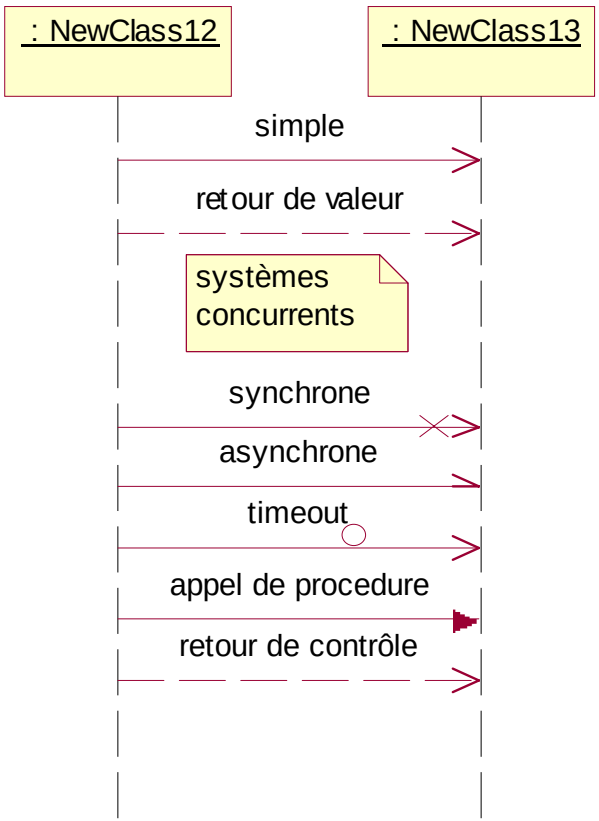
Types d'invocation (UML 1.4+)



- Systèmes non concurrents
  - Envoi de messade
  - Retour de valeur: optionnel
    - indication sur l'appel
- Systèmes concurrents
  - Invocation asynchrone
  - Retour contrôle: nécessaire
    - Indique quand l'appel retourne un résultat.

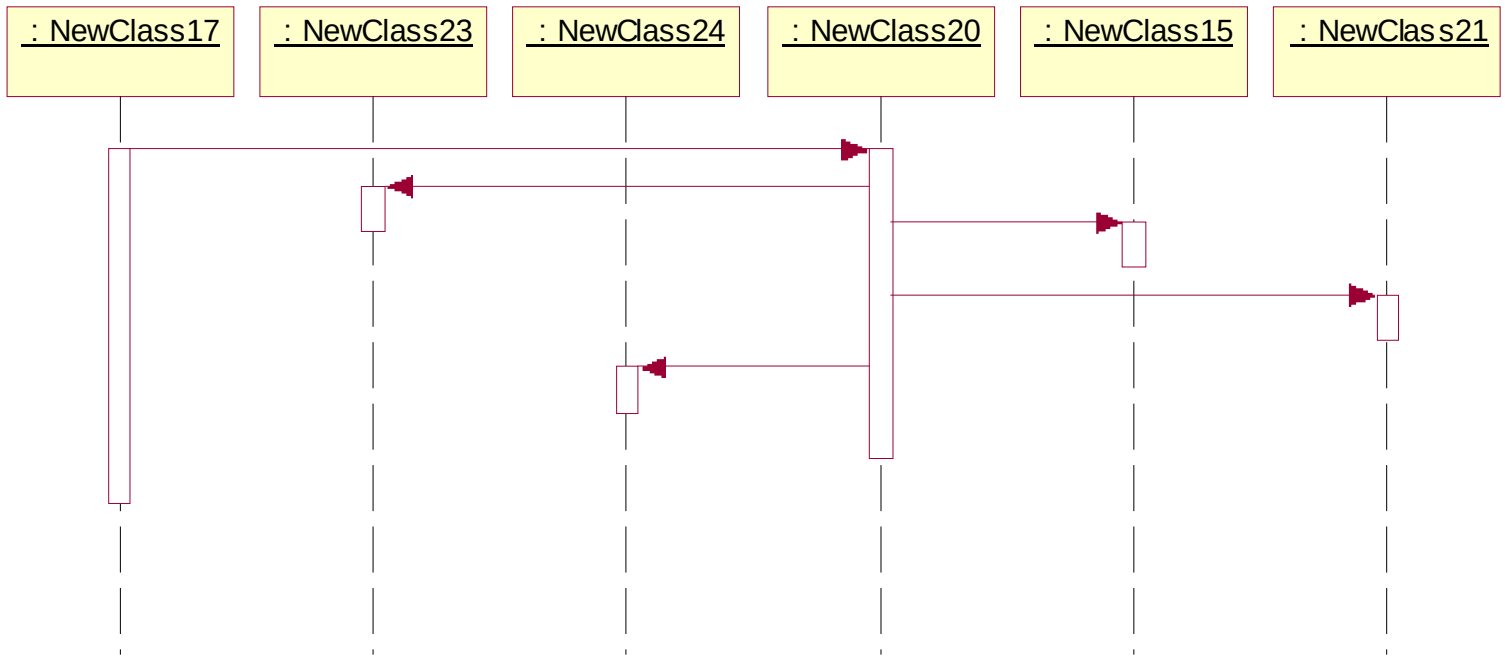
Simplification de la notation de UML1.3 et précédents

Types d'invocation (< UML 1.4)

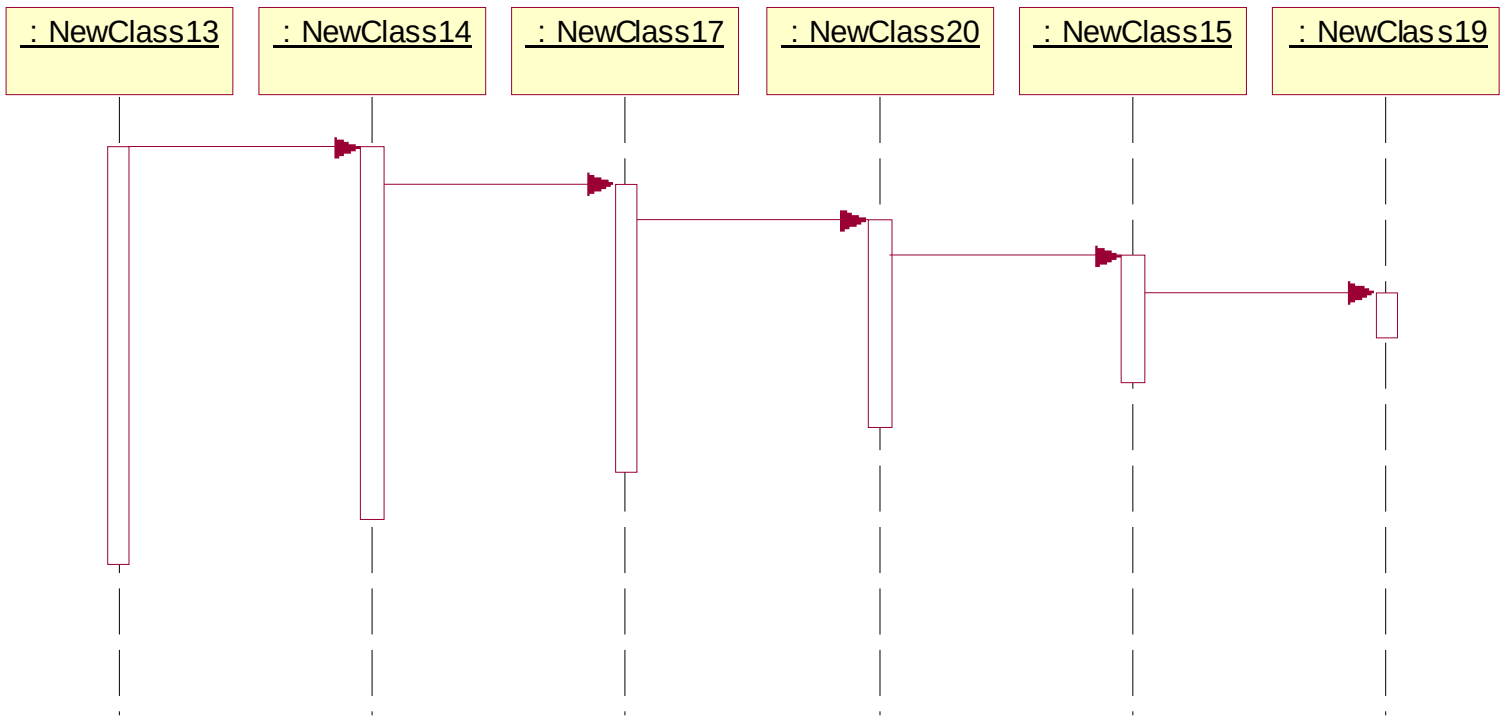


- Systèmes non concurrents
  - Envoi de message simple
    - Retour de valeur: optionnel
- Systèmes concurrents
  - Invocation synchrone
  - Invocation asynchrone
  - Invocation avec délai de prise en charge
  - Appel de procedure
  - Retour explicite du contrôle

Contrôle centralisé

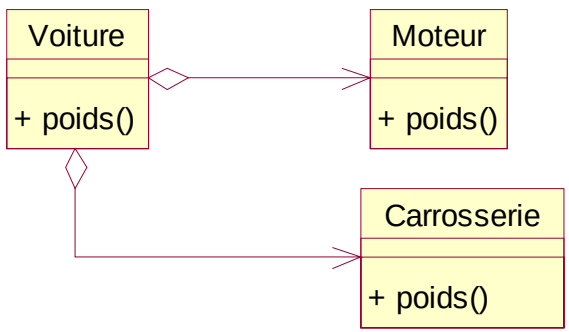


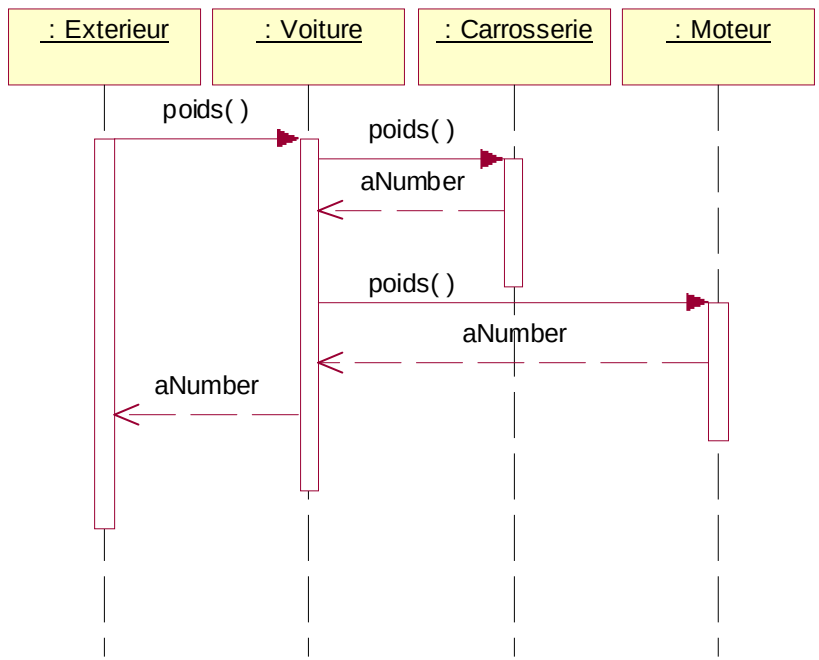
Contrôle distribué



Exemple de sequence diagram

- Le poids de la voiture est calculé à partir du poids de ses composants

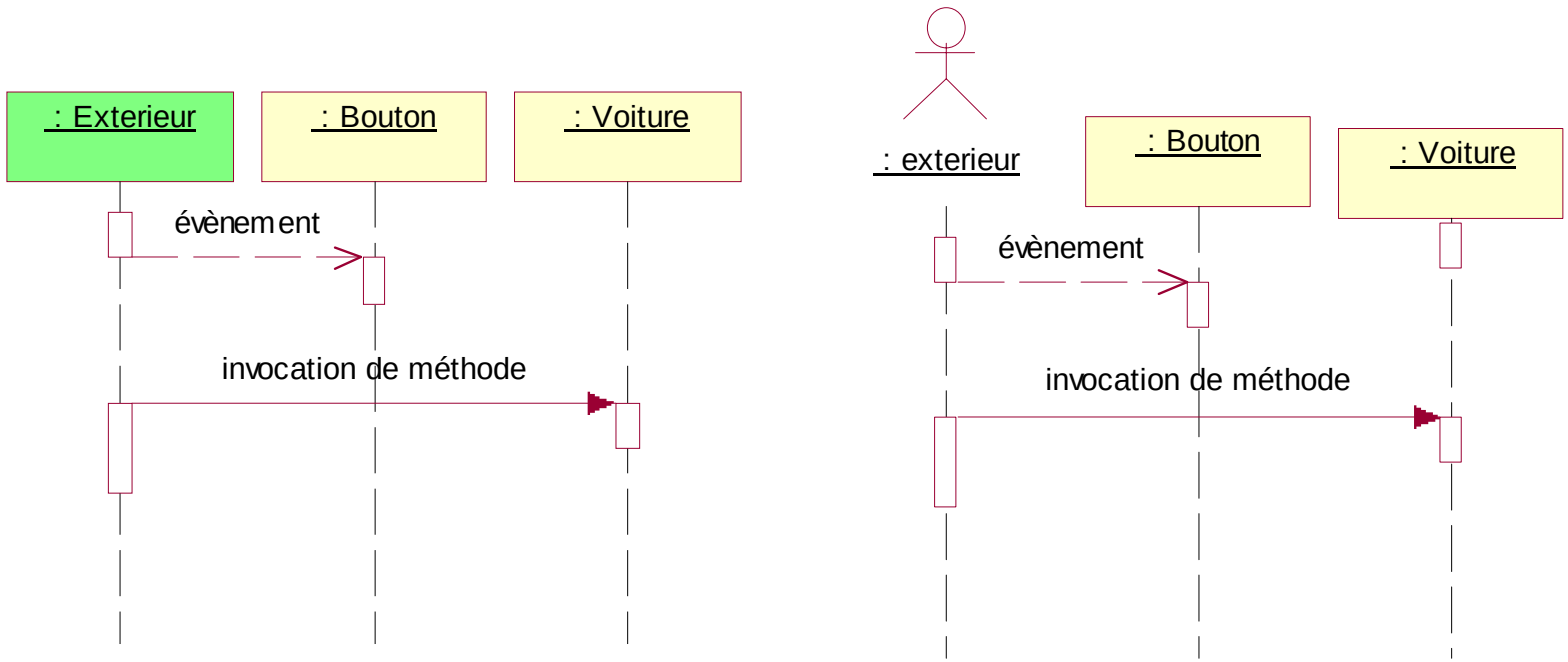




Remarque: origine d'un traitement

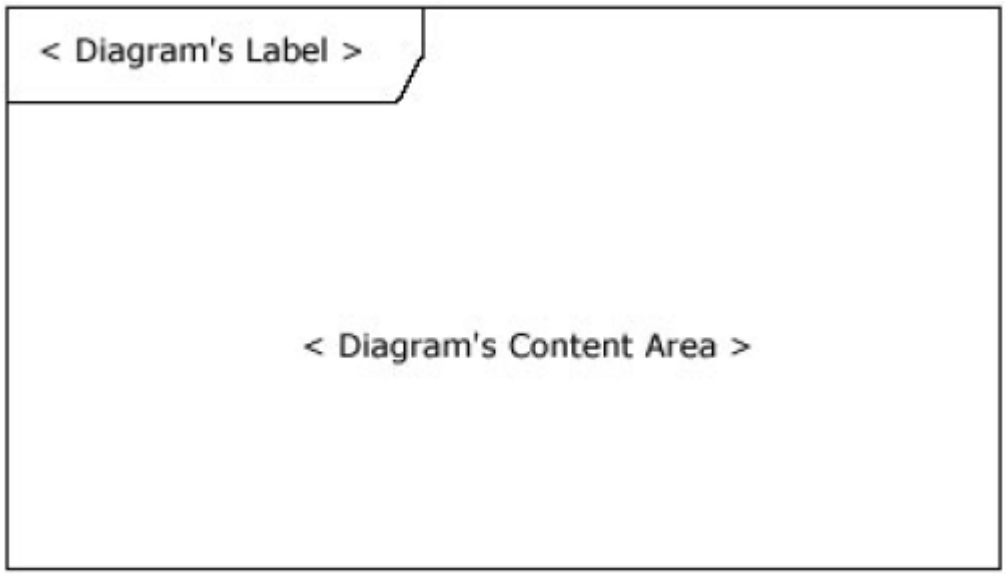
- Pour représenter le message ou l'événement déclenchant un traitement, nous avons besoin de l'objet à l'origine du message.
- Alternative
  - Représenter un objet « acteur » (utilisé pour sous-systèmes)
  - Représenter une instance identifiée comme «l'extérieur »

Exemple: origine du traitement (UML<2.0)



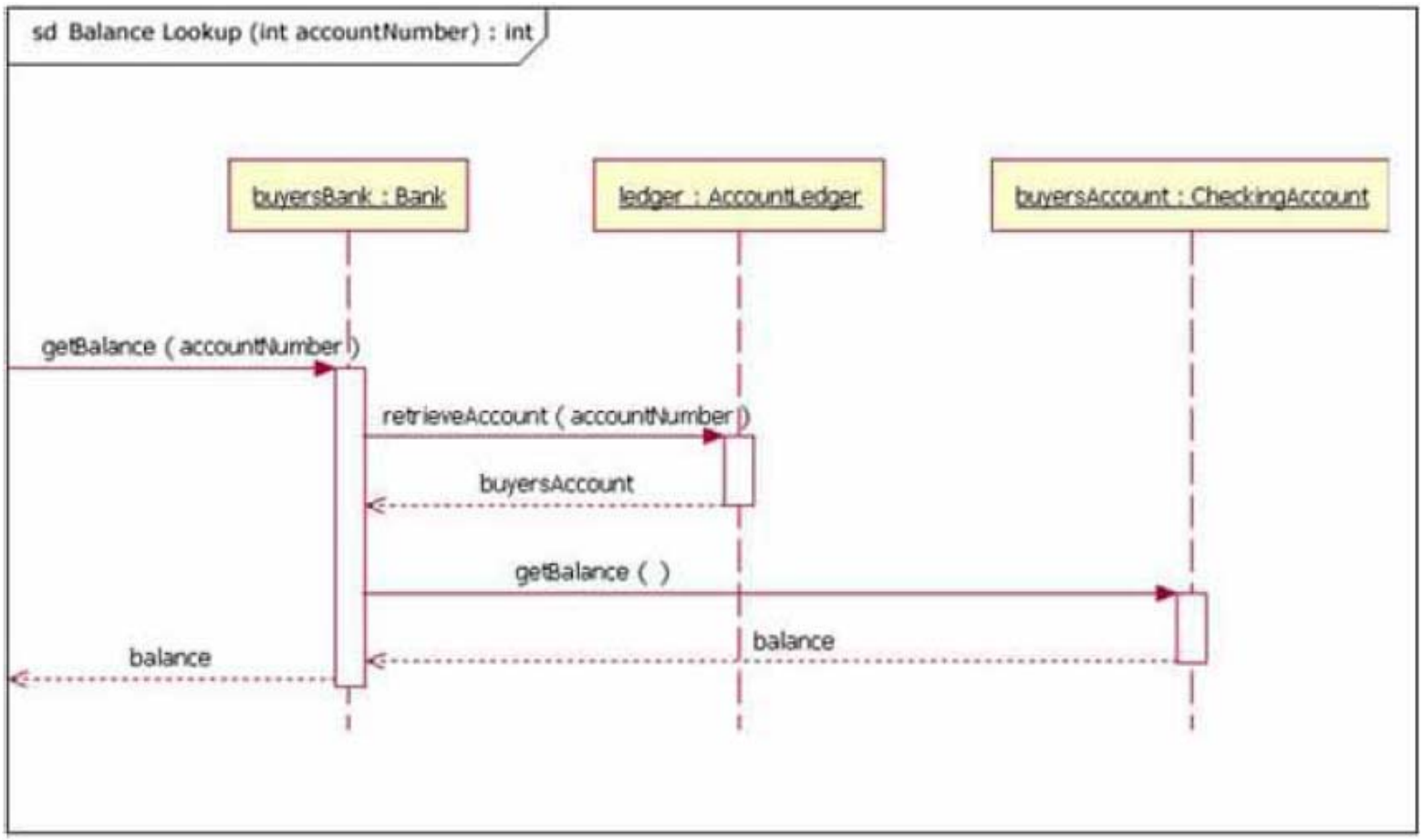
Suggestion: utiliser l'acteur pour l'intéraction avec les sous-systèmes et les intéraction avec les utilisateurs. Utiliser une instance « exterieur » pour les sous-diagrammes.
NB: le problème de la modèlisation de « l'extérieur » étant connu, UML 2.0 a défini une solution.

New in UML 2.0: Frames



The frame is used to design sub diagrams, conditionals, loops,…

Frame as « gate » in UML 2.0



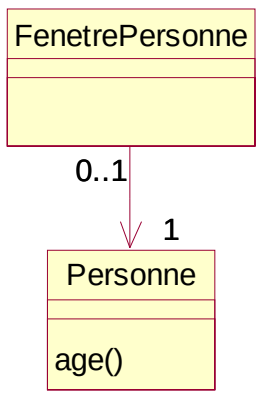
Granularité des diagrammes

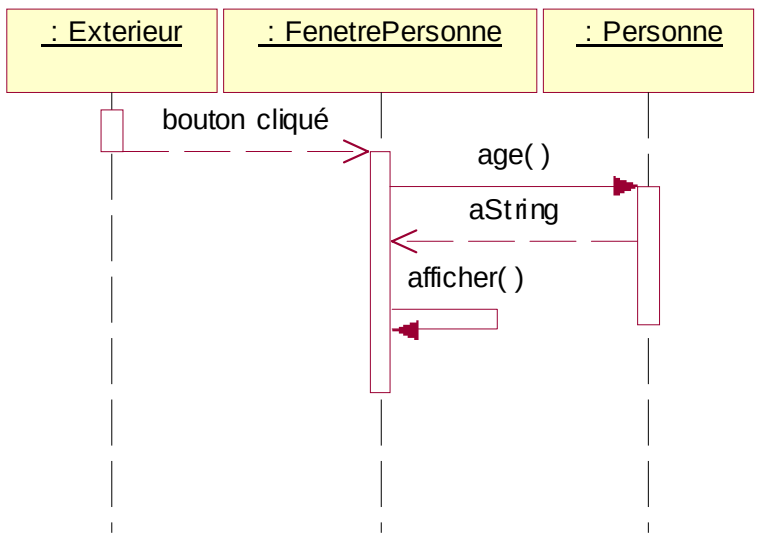
- En général, on ne représente qu'un sous-ensemble des échanges liés à la réalisation d'une fonction du logiciel.
- On exploite la propriété des composants et sous-systèmes, de se comporter, vu de l'extérieur, comme un objet.
  - On ne représente pas le fonctionnement interne des composants.
  - Ce fonctionnement peut être documenté dans un diagramme spécifique
- En résumé, dans un diagramme, il faut conserver le même niveau de granularité pour les différents messages et événements.

Exemple: granularité I

- Affichage de l'âge d'une personne en cliquant sur un bouton dans une fenêtre

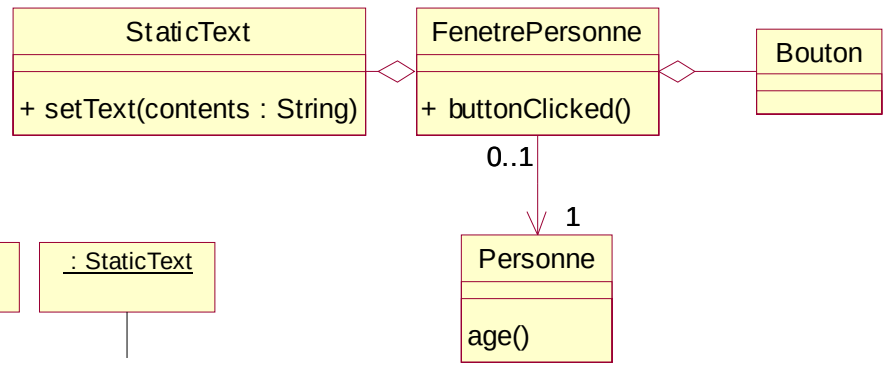
- 1- Niveau de granularité grossier

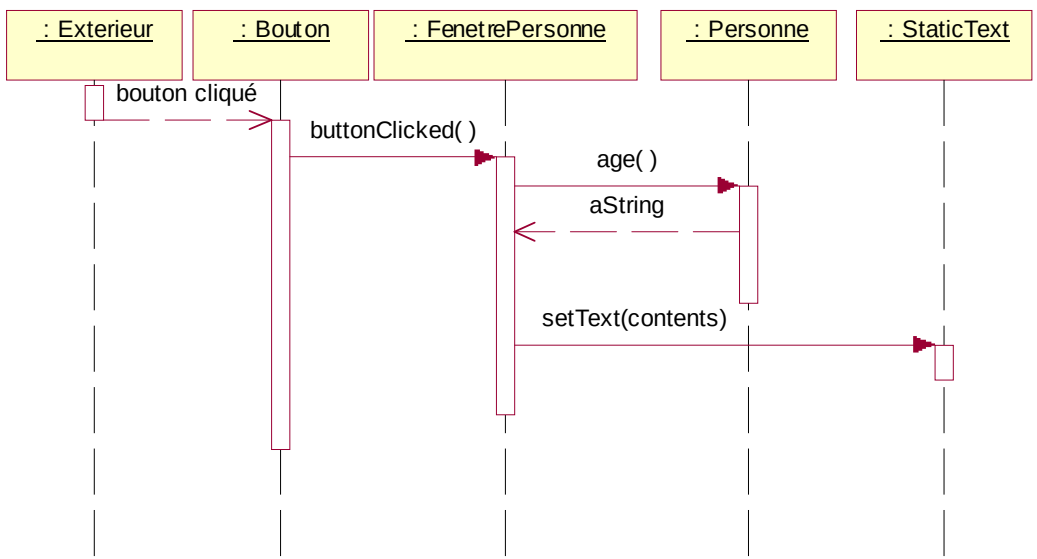




Exemple: granularité II

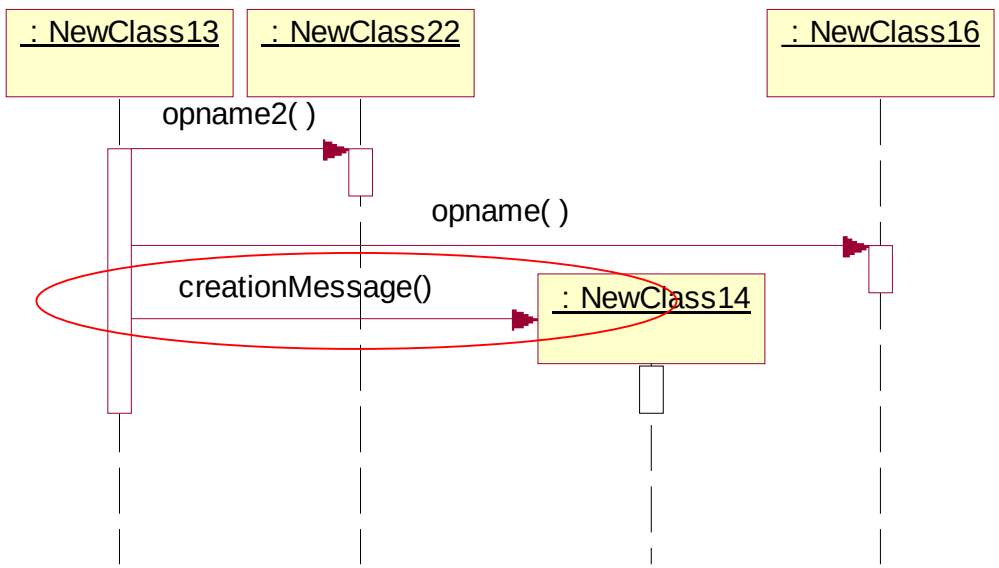
- 2- Niveau de granularité plus fin: composants de la fenêtre





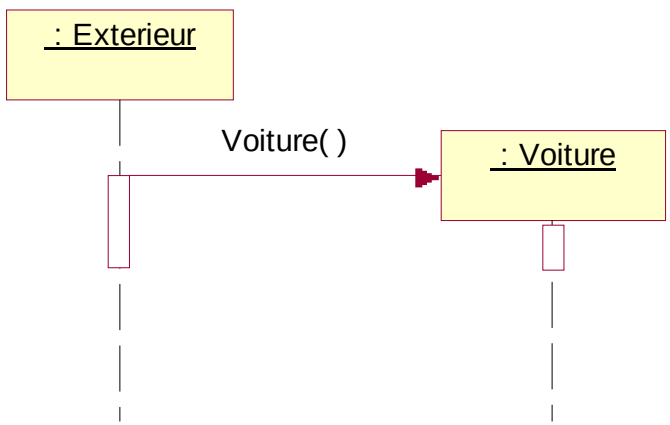
Cas particulier: création d'objet

- Message de création pointant l'objet
- Placer verticalement l'objet à l'endroit correspondant à l'instant de création



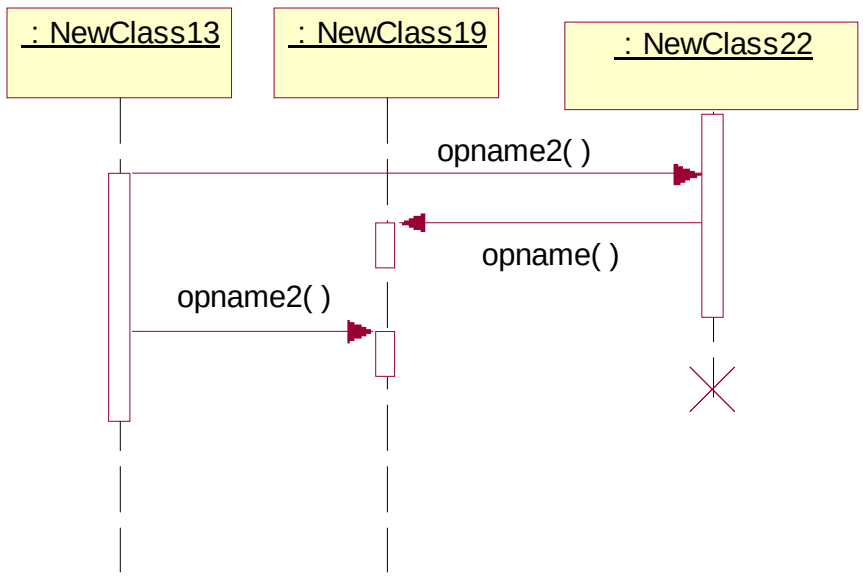
Java: constructeur

- Le message de création est représenté par le constructeur
  - public NomClasse(*paramètres d'initialisation*)



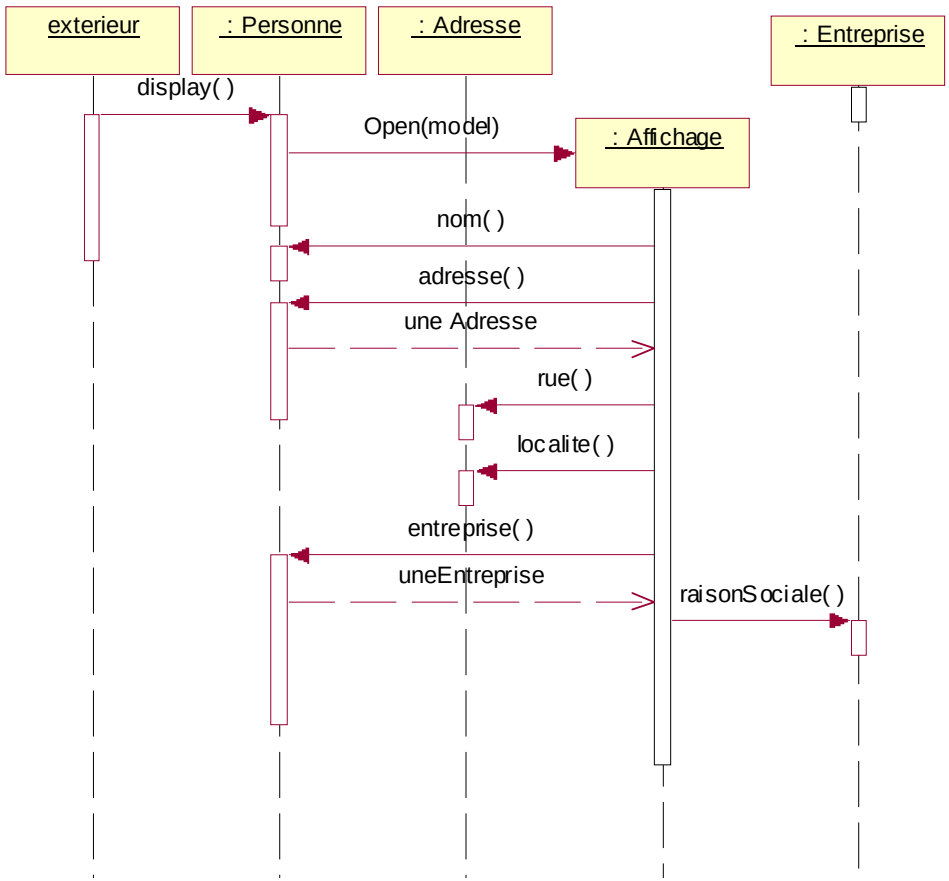
Cas particulier: destruction d'objet

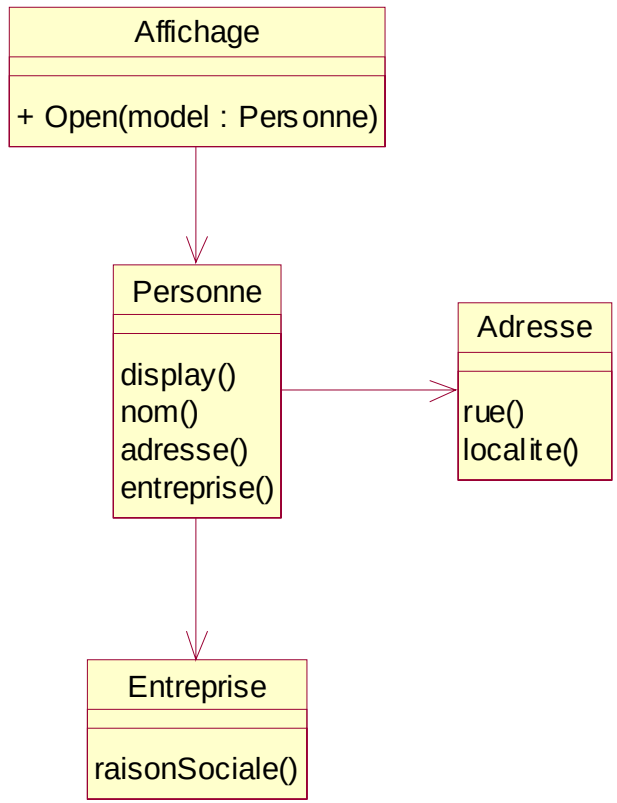
- Croix à l'extrêmité du focus of control d'une instance



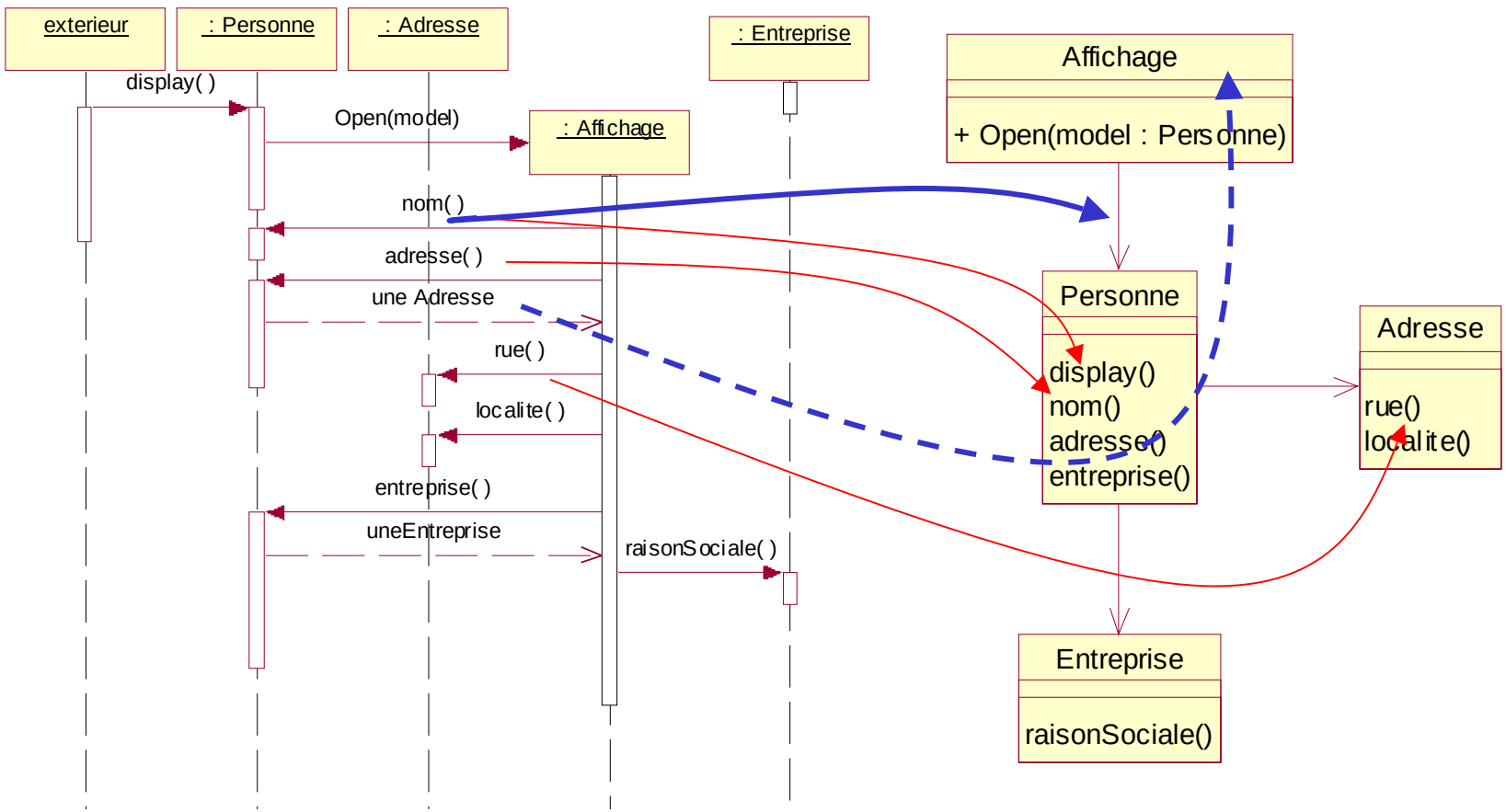
L'instance de NewClass22 est donc dé-référencée au retour de *opname()*

Exemple: lecture d'information et affichage





Interactions : nécessaire connaissance de l'instance



Points clés

- Les instances ne peuvent communiquer que si elles se « connaissent »
  - Il existe une association entre leurs classes (collaboration)
  - Un objet est retourné comme valeur de l'invocation d'un message
    - La valeur retournée est indiqué par une flèche traitillée

- Les messages envoyés doivent faire partie du protocole visible par l'objet qui invoque la méthode.

Granularité des diagrammes

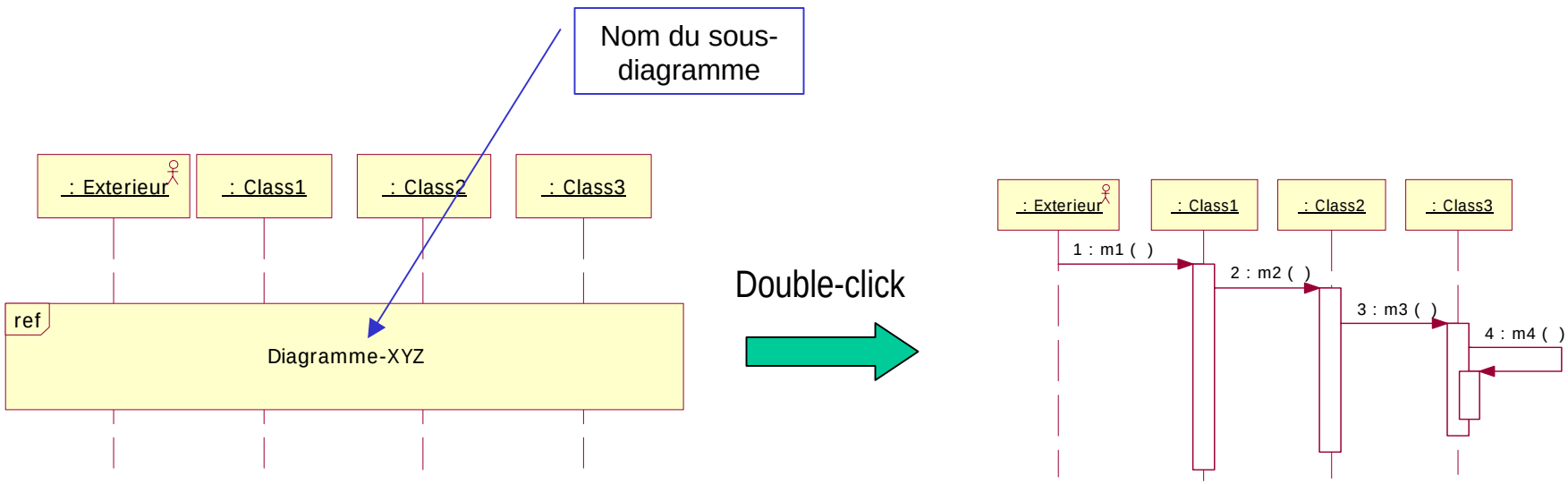
- En général, on ne représente qu'un sous-ensemble des échanges liés à la réalisation d'une fonction du logiciel.

- On exploite la propriété des composants et sous-systèmes, de se comporter, vu de l'extérieur, comme un objet.
  - On ne représente pas le fonctionnement interne des composants.
  - Ce fonctionnement peut être documenté dans un diagramme spécifique

- Dans un diagramme, il faut conserver le même niveau de granularité pour les différents messages et événements.

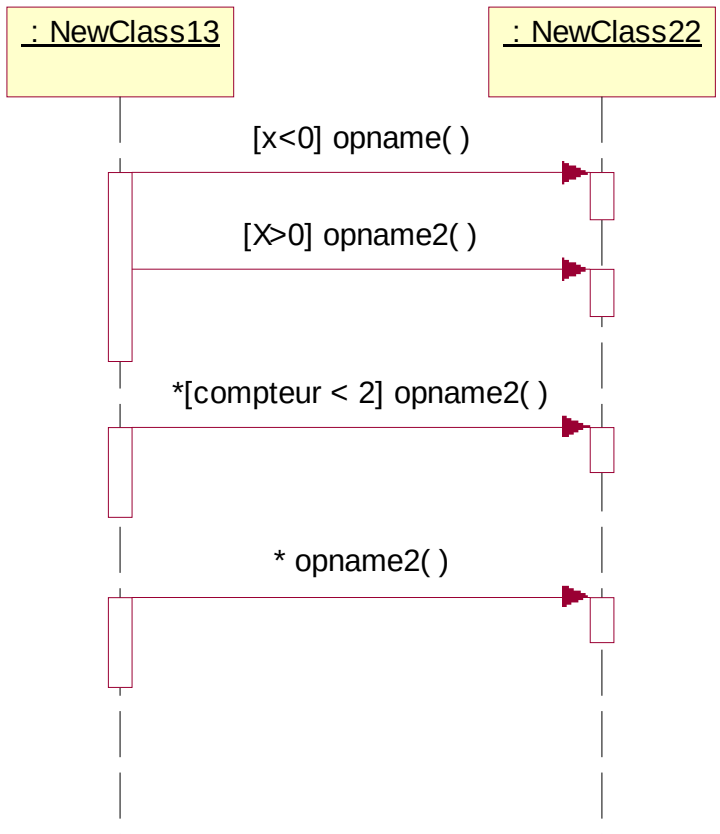
Sous-diagrammes et « Global Actions »

- Référence à un sous-diagramme via un « Frame » du type référence (tag: « **ref** »).
  - C'est le seul « frame » disponible en XDE (2003.06.00)



Ajout: par drag and drop d'un SD depuis le model explorer dans un SD ouvert

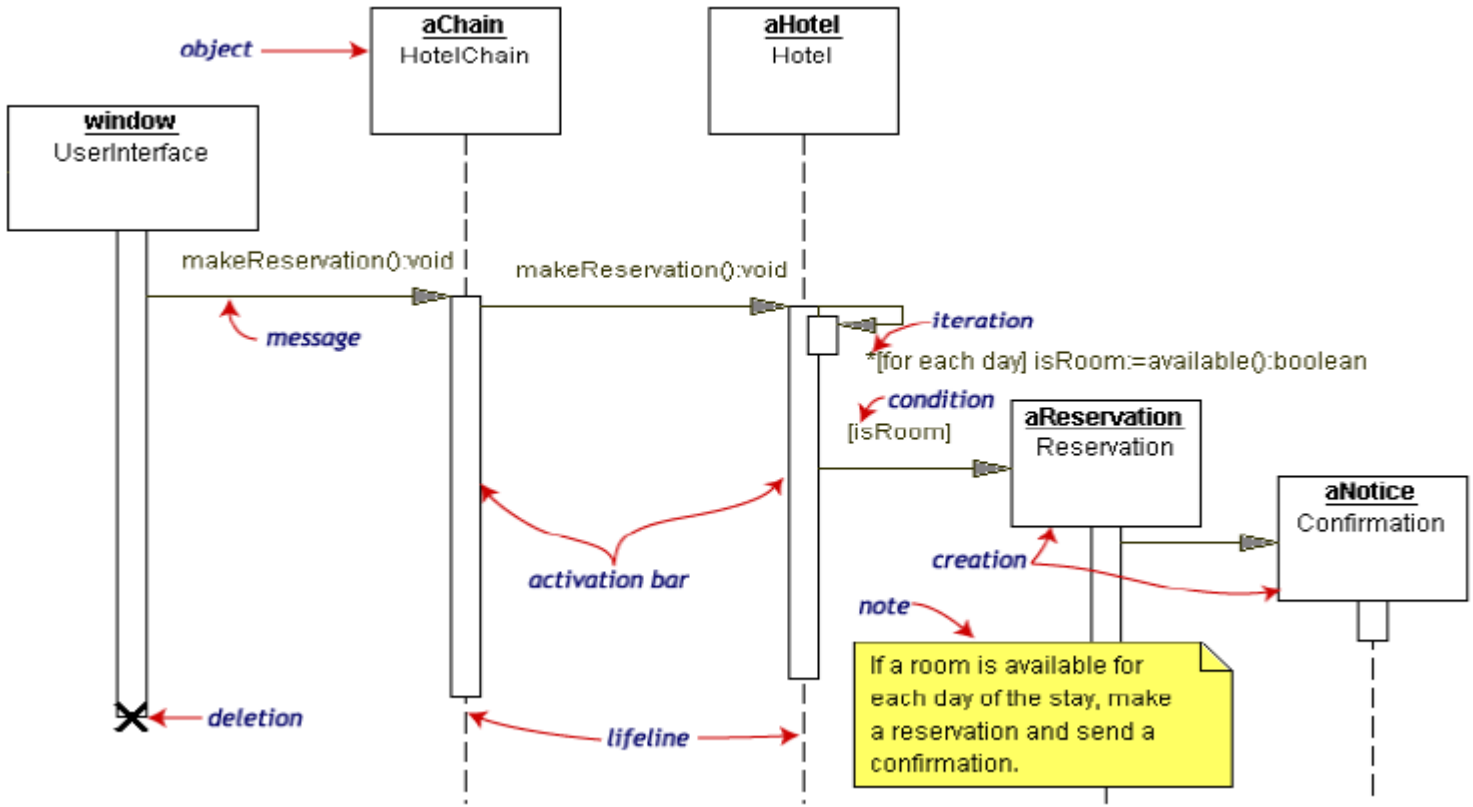
Envoi conditionnel, itération



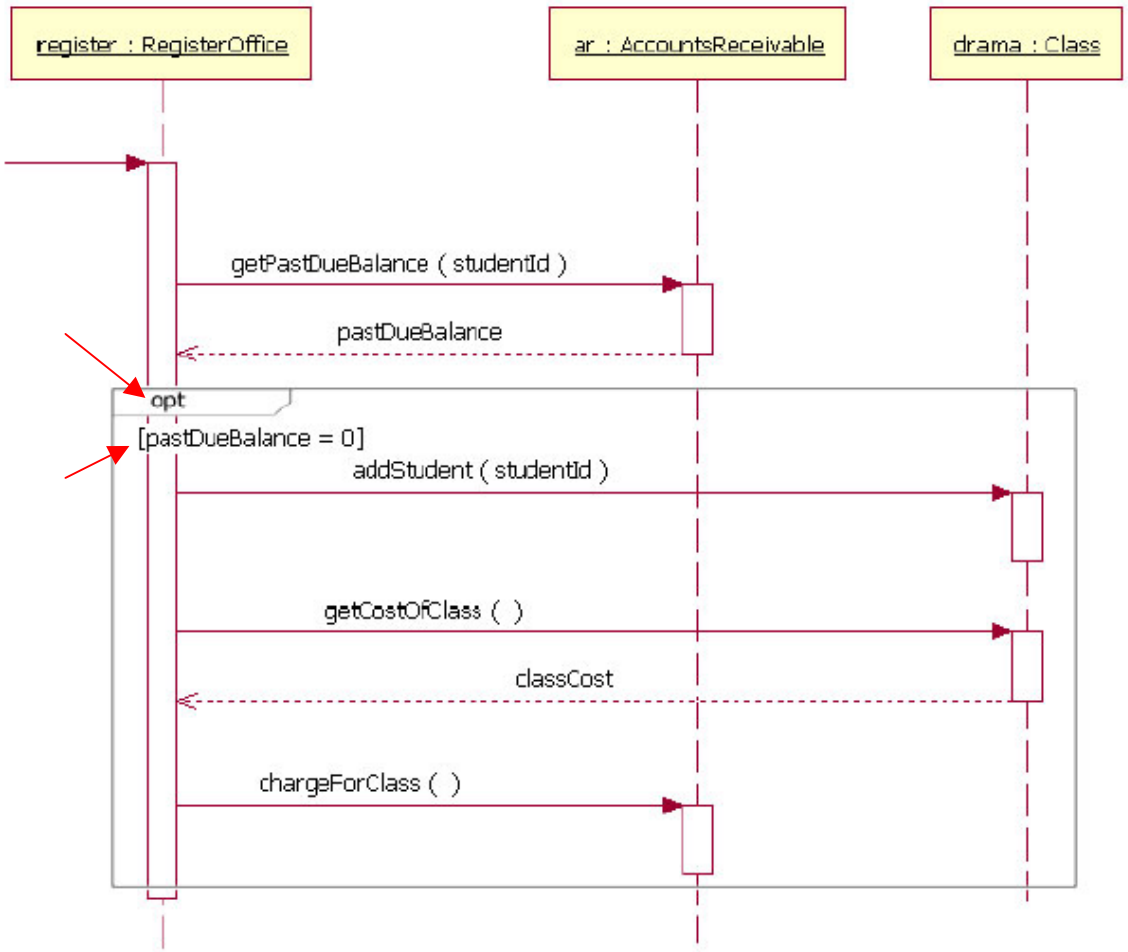
Indiquer la condition entre crochets

- Le message n'est envoyé que si la condition est satisfaite. UML ne spécifie pas le format de la condition. (pseudo-code, java,…)
- Avec « * » le message est envoyé itérativement tant que la condition est vraie.
- La condition d'itération peut être omise (non spécifiée)

Resumé

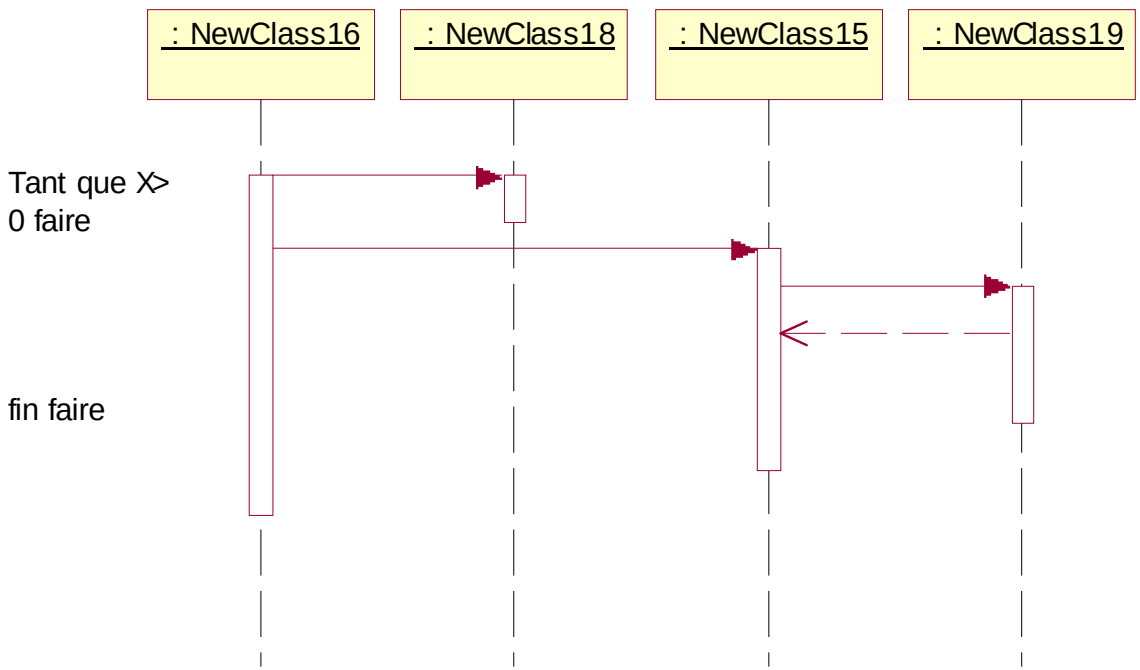


Frames pour le traitement optionnel : **opt** (UML 2.0+)

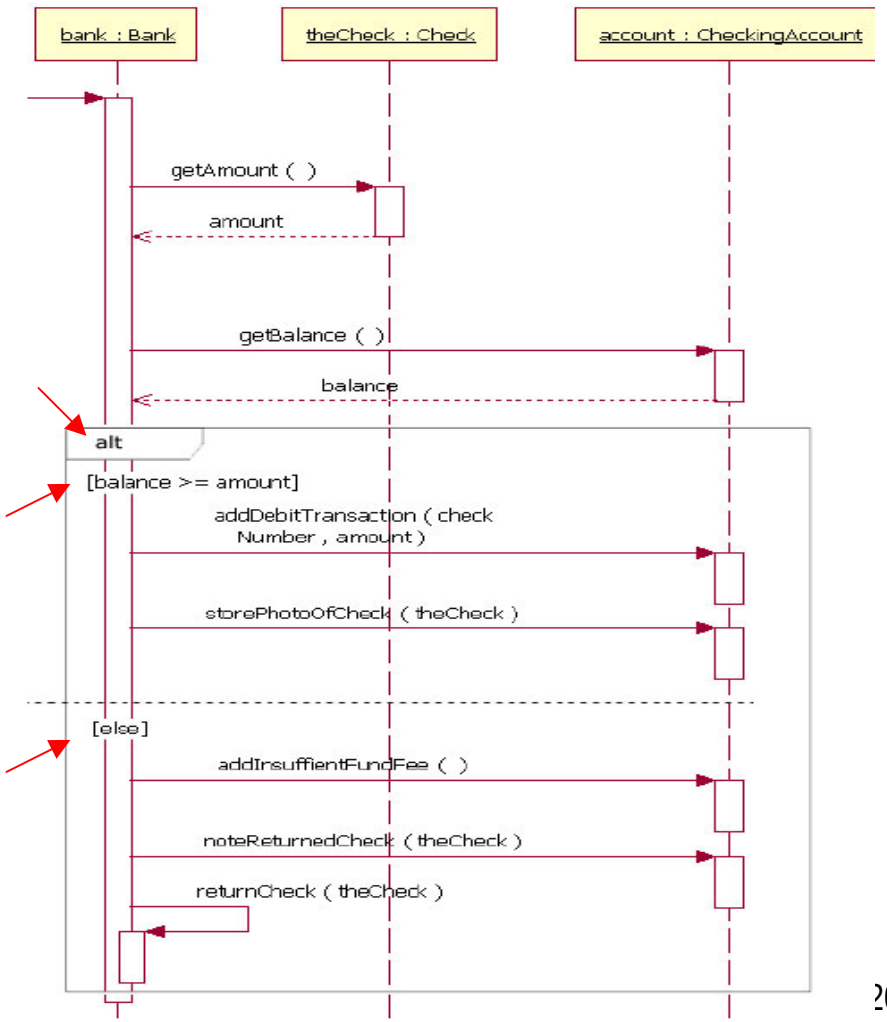


Boucles, containtes, conditions, commentaires (UML < 2.0)

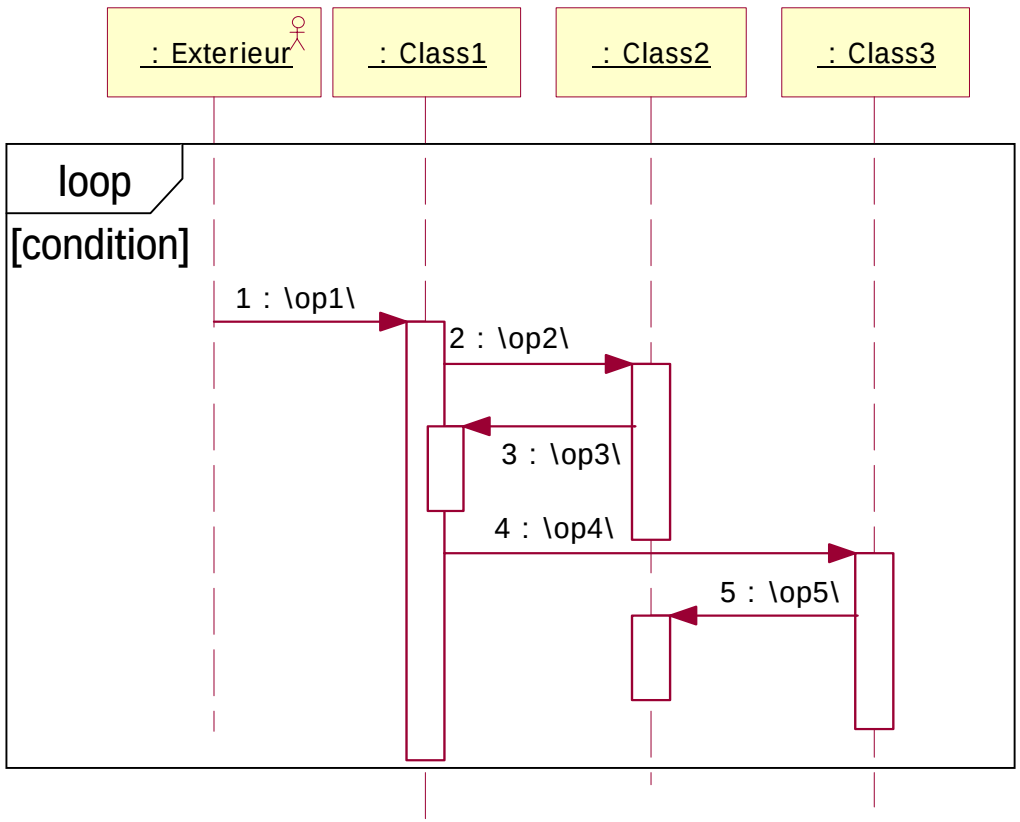
Des informations supplémentaires peuvent être exprimées dans la marge de gauche. Texte libre, pseudo-code, OCL.



Frames pour le traitement conditionnel : **alt** (UML 2.0+)

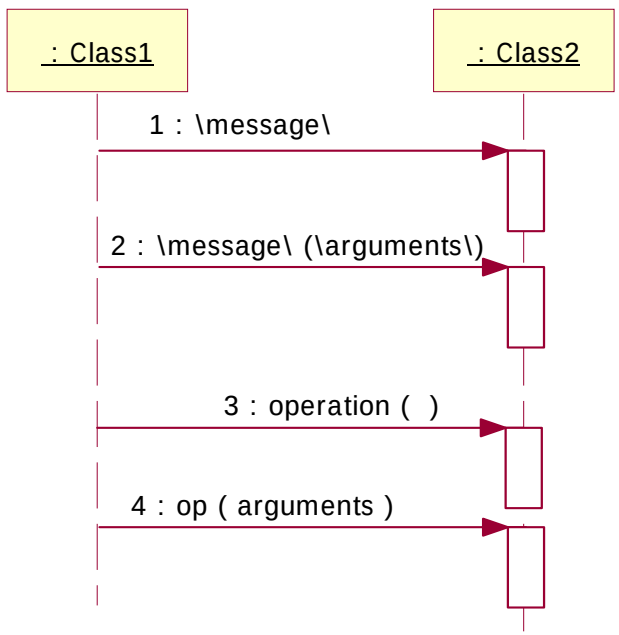


Frames pour boucles: loop (UML 2.0+)



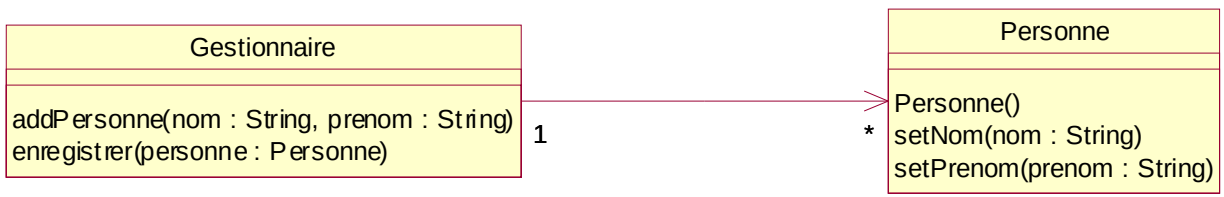
Message et opérations

- Départ: message (responsabilités) (\resp\ )
- Conception détaillée: message => opération (op())

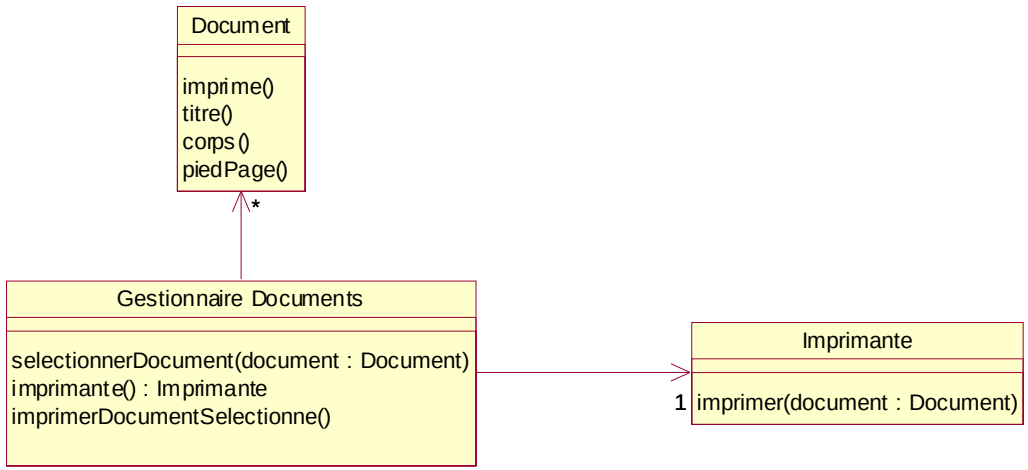


Exercice1

- Créer le séquence diagram du système ci-dessous.
  - Quand le gestionnaire reçoit le message *addPersonne(nom:String, prenom :String)* il crée une nouvelle instance de *Personne*, lui assigne ses nom et prenom puis l'enregistre par *enregistrer(personne:Personne)*

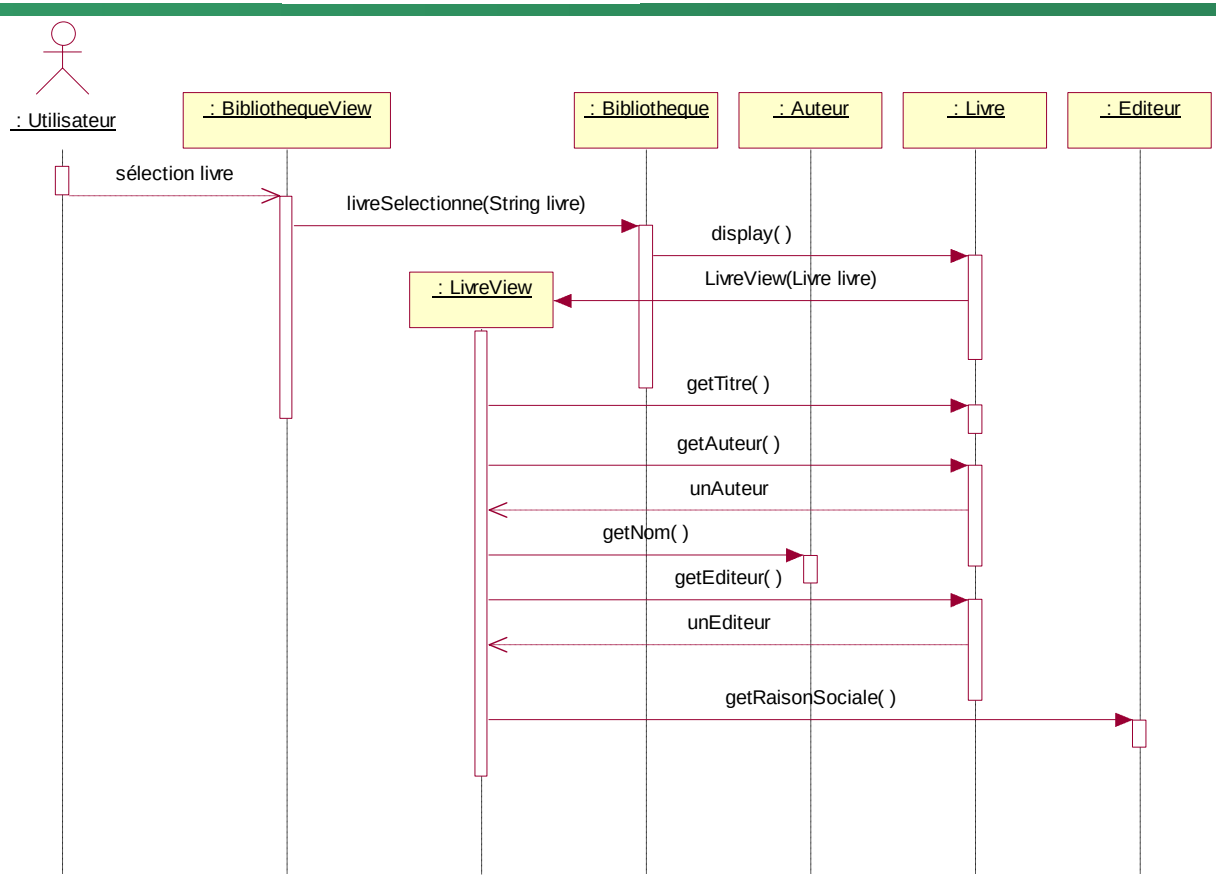


Exercice 2



Etablissez le sequence diagram de l'opération d'impression d'un document. Cette opération débute par la sélection d'un document dans le gestionnaire. Ensuite, on demande à ce même gestionnaire d'imprimer le document sélectionné. Ce dernier demande alors au document lui-même de s'imprimer. Le document demande alors au gestionnaire de lui retourner l'imprimante courante puis communique avec cette imprimante pour lui demande de l'imprimer. Finalement, l'imprimante lui demande successivement ton titre, le corps du texte et le pied de page.

Exercice3



Représentez le diagramme de classe compatible avec ce diagramme de séquence.

Diagramme de collaboration

- Même information que le sequence diagram
  - Représentation des messages sur la base de la structure du modèle objet
  - Représentation du temps par un numéro de séquence

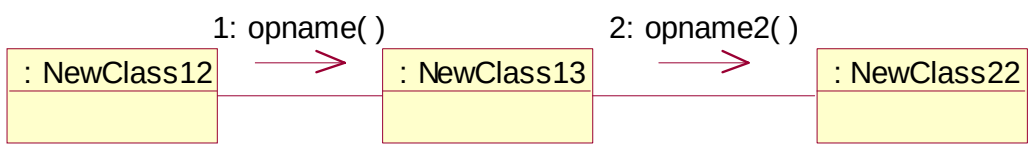
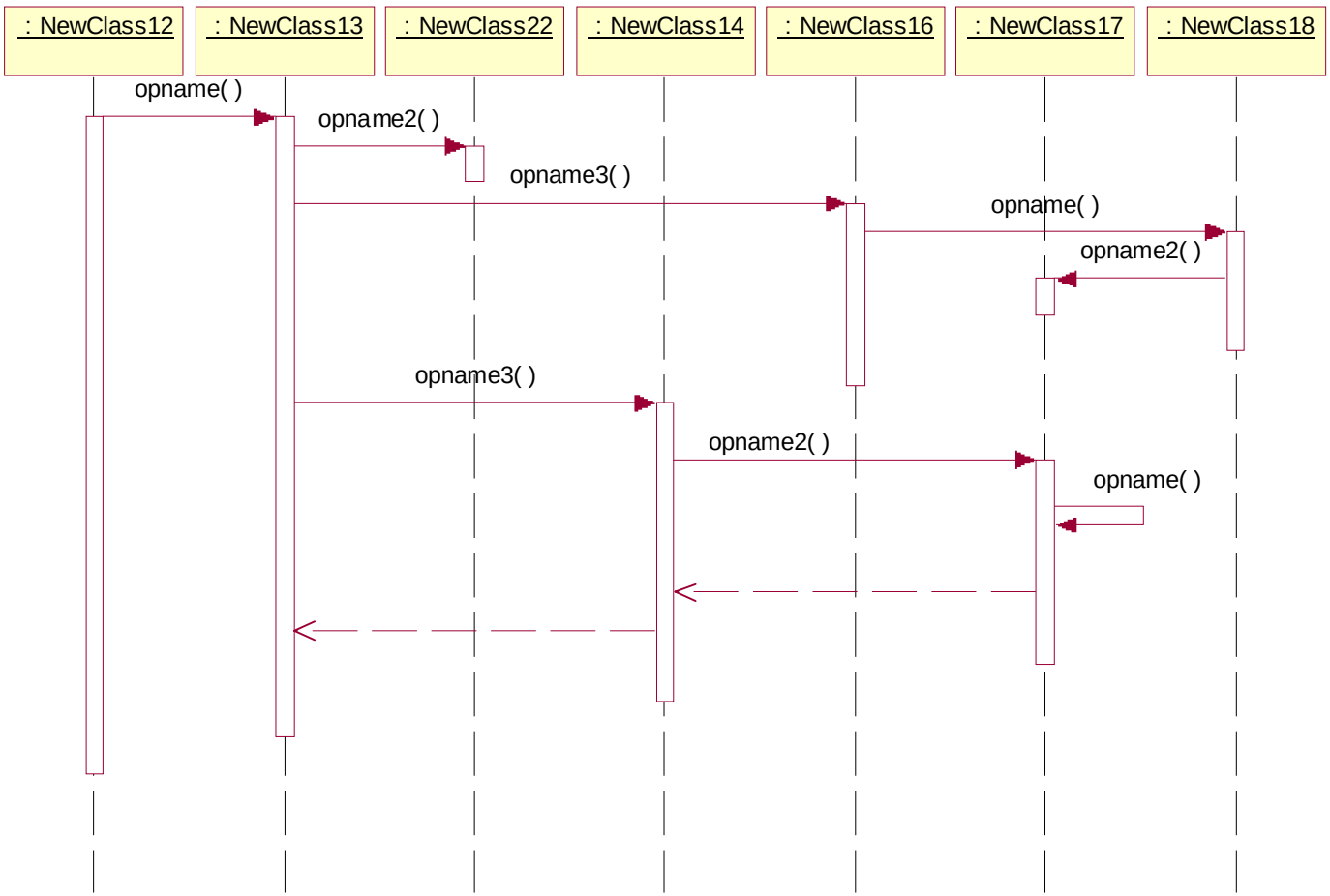
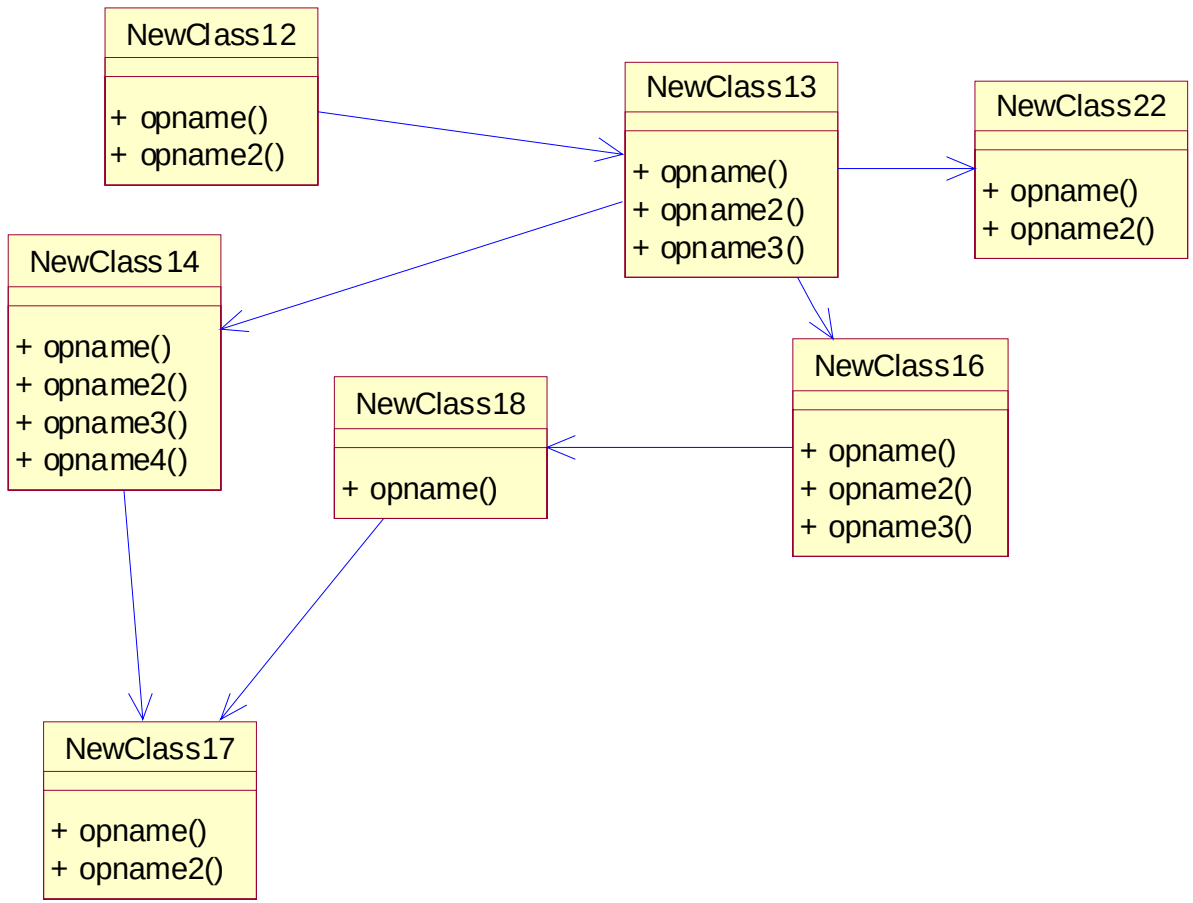


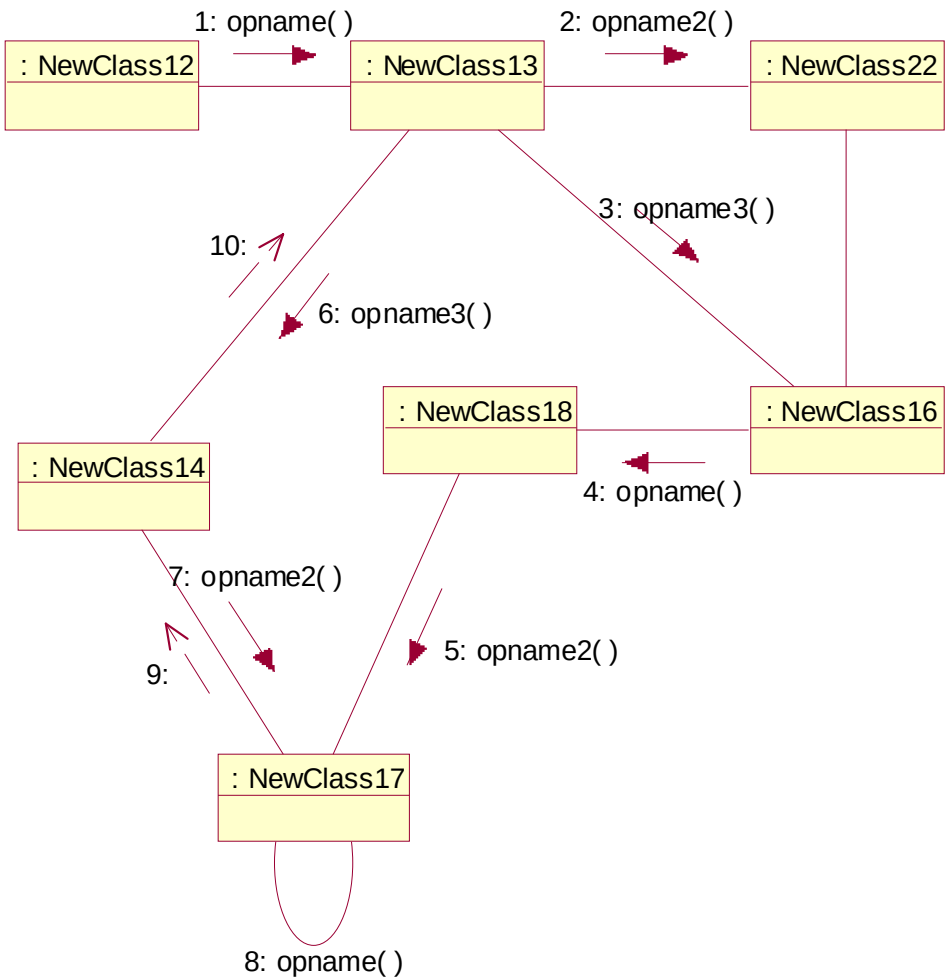
Diagramme de collaboration : dynamique des instances….



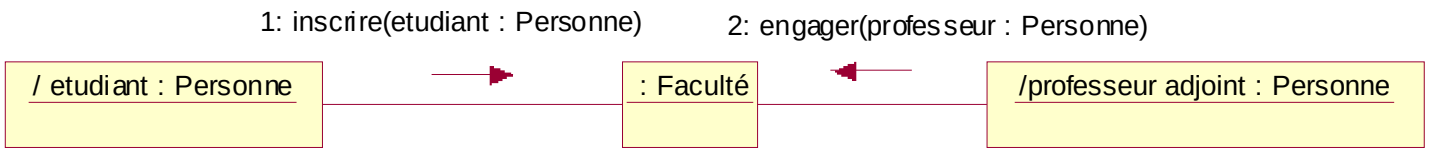
...représentée selon l'architecture des classes

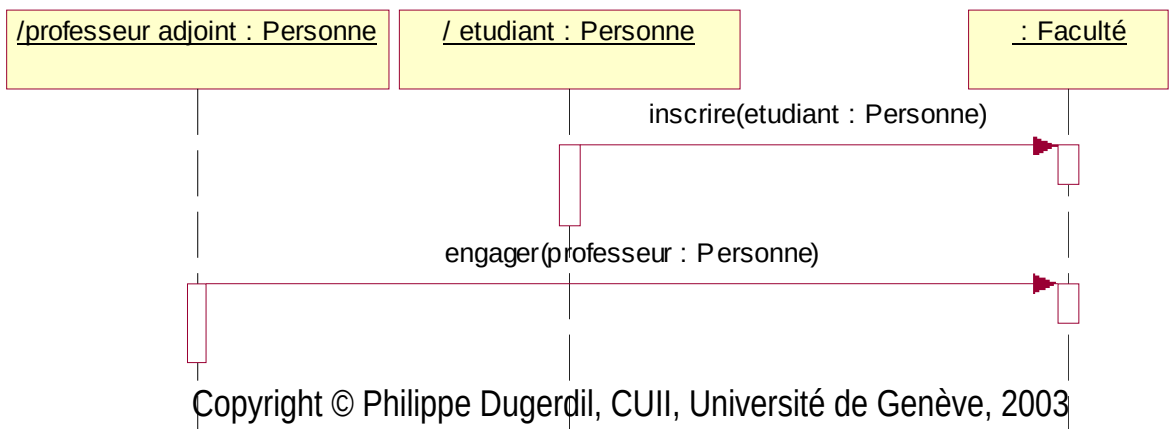


Résultat: ensemble d'instances qui collaborent



Cas des objets à rôles multiples: spécification du rôle

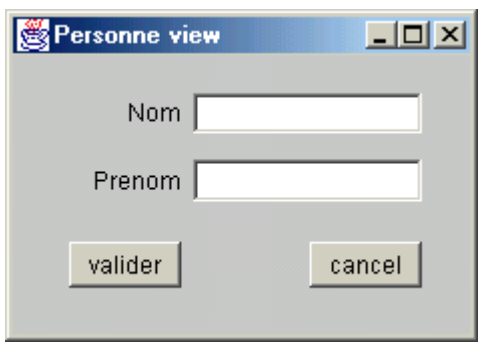


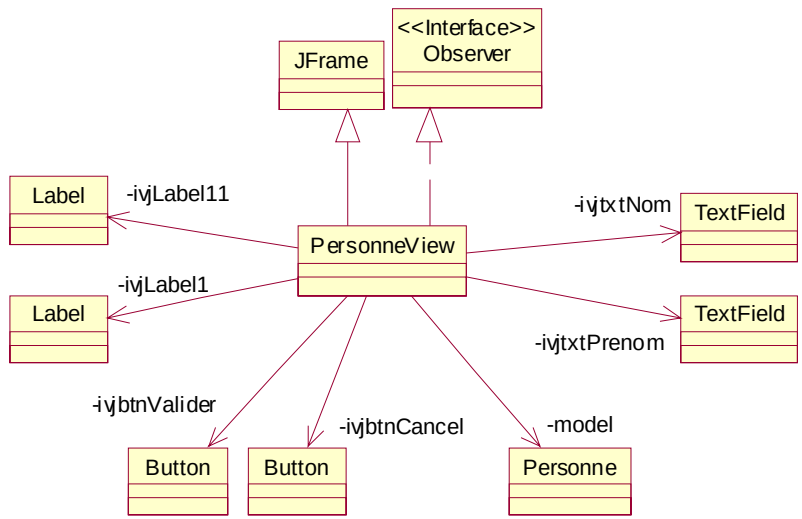


Exemple: utilisation des rôles

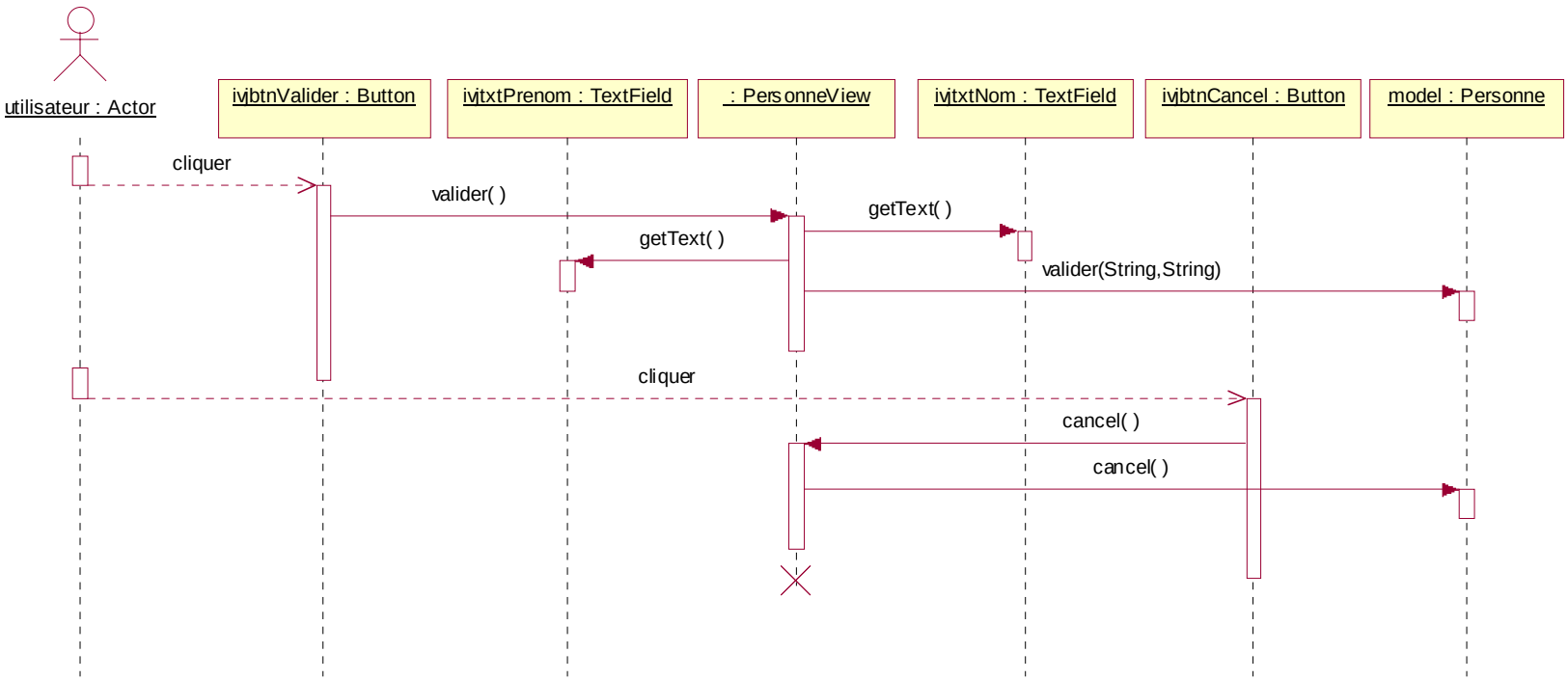
- Si plusieurs instances distinctes de la même classe interagissent dans un séquence diagram ou un collaboration diagram, on les distingue :
  - par leur rôle
  - par leur nom (souvent: le nom de la variable qui la référence)

- Exemple: identification par le nom de la variable

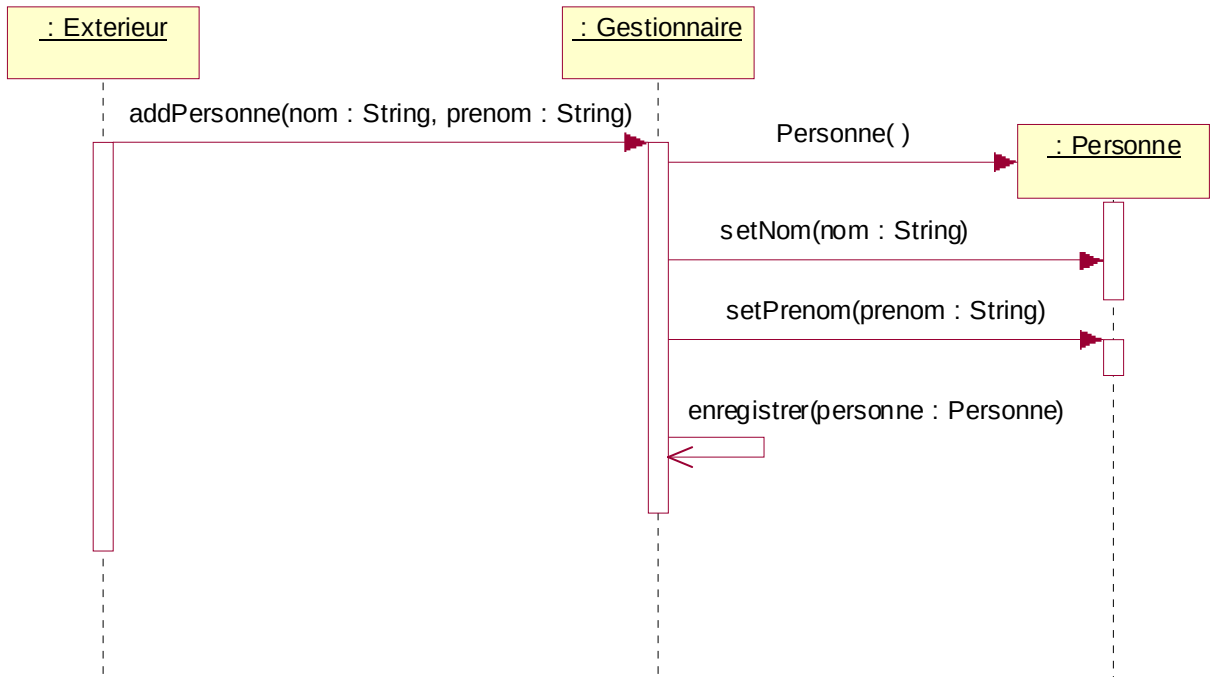




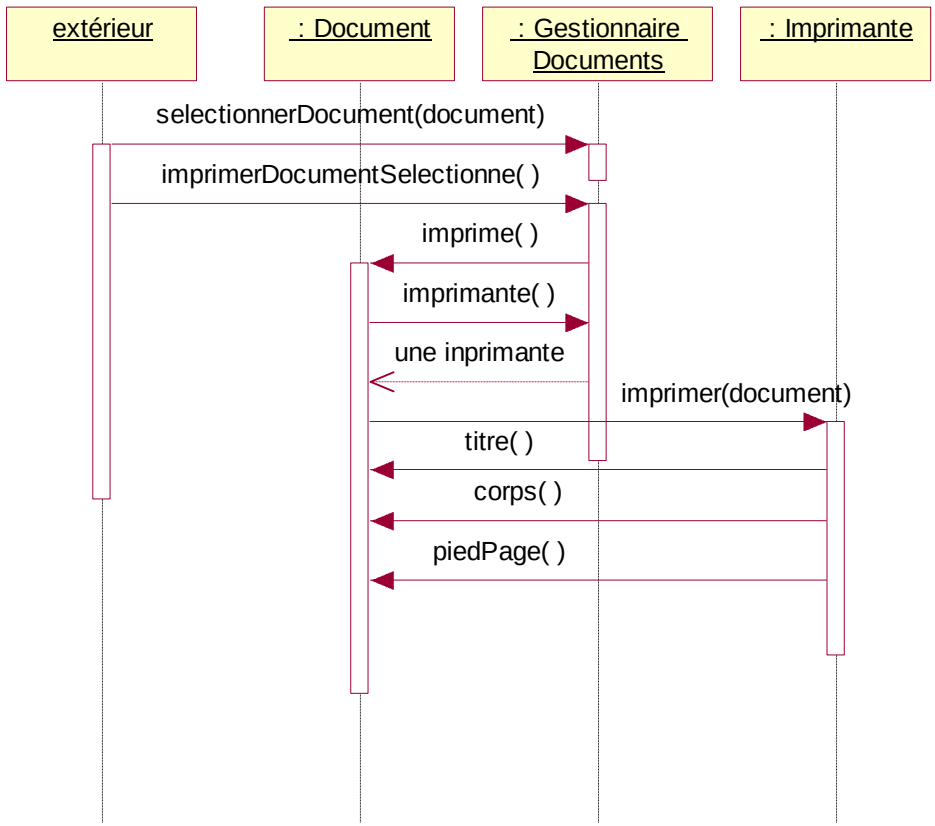
Sequence diagram de PersonneView



Corrigé 1



Corrigé 2



Corrigé 3