

## R3.04 Qualité de développement

I.Borne  
(isabelle.borne@univ-ubs.fr)

IB- R3.04

1/69

La ressource R3.04 :

Objectif : approfondir la qualité de développement

Approfondissement :

- des concepts de développement
- des modèles de conception

Conception et mise en œuvre

- patrons de conception
- éléments d'architecture logicielle
- restructuration de code

Mettre en œuvre des bonnes pratiques de programmation par objets

IB- R3.04

2/69

M3105

COURS n°1

I.Borne

- ♦ Quelques mots sur le processus unifié
- ♦ Particularités des diagrammes de classes
- ♦ Modélisation des interactions
  - ♦ Diagramme de séquence
  - ♦ Diagramme de communication
- ♦ Passer de la conception au code

IB- R3.04

3/69

Rappel des termes autour du concept de modèle

- ♦ **Modèle** : représentation abstraite d'une spécification, d'une conception ou d'un système selon un point de vue particulier
- ♦ **Processus** : tâche en train de s'exécuter
  - ♦ A faire : rechercher les définitions de processus métier et la gestion de production selon la norme ISO 9001.
- ♦ **Méthode** : savoir-faire développé par une personne ou une équipe dans un domaine
- ♦ **Méthodologie (de conception)** : l'ensemble des méthodes de conception
- ♦ **Processus de développement** : ensemble de règles qui définit comment le développement doit être réalisé.

IB- R3.04

4/69

## Processus unifié (UP Unified Process)

- ♦ Il s'agit d'une méthode prise en charge du cycle de vie d'un logiciel et donc du développement des logiciels à objets.
- ♦ Un processus unifié est un processus de développement logiciel construit sur UML.
- ♦ Tout processus unifié est
  - itératif
  - Incrémental
  - piloté par les risques
  - orienté composant
  - orienté utilisateur

IB- R3.04

5/69

Itératif : chaque itération porte sur des degrés d'abstraction de plus en plus précis, et permet de produire des traces nécessaires au contrôle de changement.

Incrémental : la définition d'incrément de réalisation est en effet une meilleure pratique de gestion des risques (technique et fonctionnel)

Un projet qui ne produit rien d'exécutable dans les 9 mois court un risque majeur d'échec.

Piloté par les risques : les causes majeures d'échec doivent être écartées. Une première cause provient de l'incapacité de l'architecture technique à répondre aux contraintes opérationnelles. Une seconde est liée à l'inadéquation du développement aux besoins des utilisateurs.

Orienté composant : constitue le support nécessaire à la réutilisation logicielle et offre des perspectives de gains non négligeables

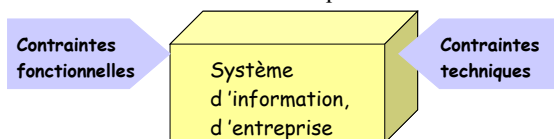
Orienté utilisateur : construit à partir des modes d'utilisation attendus par les acteurs du système

IB- R3.04

6/69

## 2TUP : « 2 Tracks Unified Process »

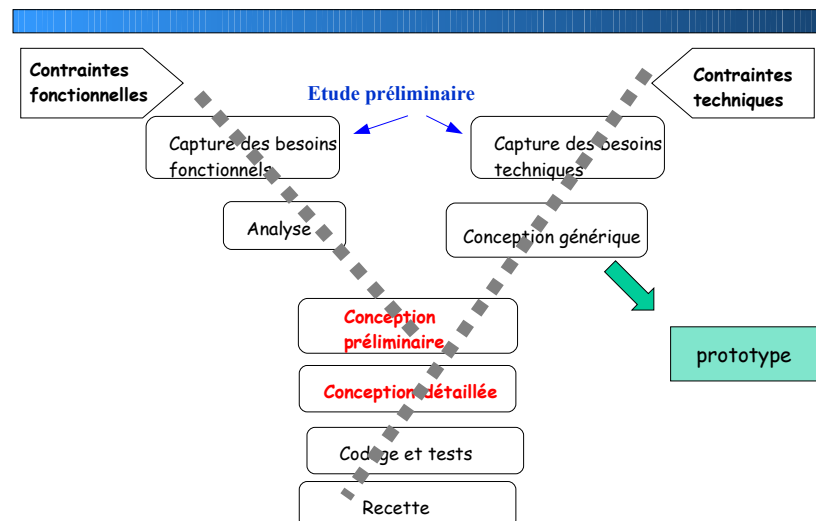
- ♦ 2TUP : processus de développement logiciel qui implémente le processus unifié.
- ♦ Il apporte une réponse aux contraintes de changement continu imposées aux SI d'entreprise.
- ♦ Le processus suit deux chemins (2 tracks) :
  - le chemin fonctionnel
  - le chemin d'architecture technique



IB- R3.04

7/69

## Le processus de développement en Y



IB- R3.04

8/69

## Architecture technique

- ♦ **Capture des besoins techniques** : recense toutes les contraintes et les choix qui dimensionnent la conception du système (outils, matériels, contraintes d'intégration avec l'existant)
- ♦ **Conception générique** : définit les composants nécessaires à la construction de l'architecture technique (indépendante des aspects fonctionnels). Elle construit le squelette du système informatique que l'on valide avec la réalisation d'un prototype.
- ♦ **Conception préliminaire** : étape délicate car elle intègre le modèle d'analyse dans l'architecture technique de manière à tracer la cartographie des composants du système à développer
- ♦ **Conception détaillée** : étudie comment réaliser chaque composant
- ♦ **Codage et tests** : produit les composants au fur et à mesure.
- ♦ **Recette** : valide les fonctions du système développé.

IB- R3.04

9/69

## Etude préliminaire

Un document doit être rédigé contenant les paragraphes suivants :

- ♦ **Présentation du projet** : Il s'agit de reformuler le besoin tel qu'il a été exprimé dans le cahier des charges.
- ♦ **Définition des grands choix techniques** : matériels, logiciels, langages, architecture, méthodes utilisées.
- ♦ **Recueil des besoins fonctionnels** : recenser les traitements informatiques prévus.
- ♦ **Recueil des besoins opérationnels** : expliciter les volumes et la définition de la sécurité
- ♦ **Acteurs du système** : les acteurs et précise les prérogatives de chacun
- ♦ **Diagrammes de contexte**
  - un diagramme de contexte statique
  - un diagramme de contexte dynamique

IB- R3.04

10/69

## Conception préliminaire

Elle explique comment organiser un modèle de conception au vu des regroupements d'analyse et des couches logicielles d'architecture.

En entrée : les acteurs et les modèles d'analyse

1. Concevoir le déploiement
2. Concevoir le modèle d'exploitation
3. Organiser le modèle logique de conception
4. Concevoir les interfaces
5. Concevoir la structure de la présentation
6. Organiser la configuration logicielle

IB- R3.04

11/69

## Conception détaillée

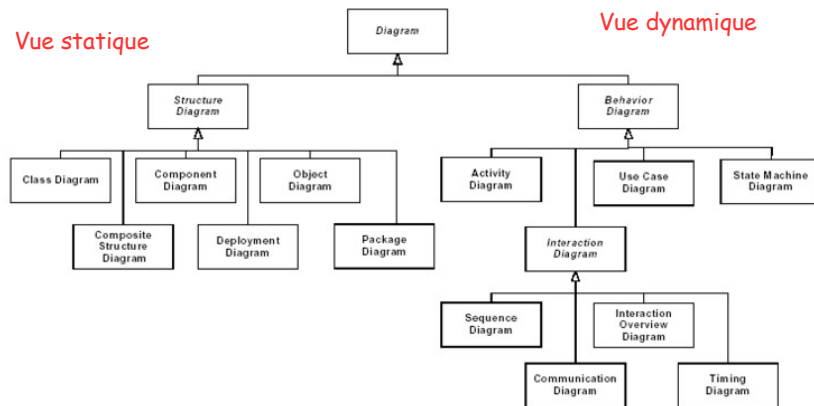
La conception détaillée illustre la modélisation des solutions informatiques en appliquant différents patrons de conception, suivant les couches que l'on désire réaliser

1. Concevoir le modèle logique
  1. Concevoir les classes
  2. Concevoir les associations
  3. Concevoir les attributs
  4. Concevoir les opérations
  5. Valider le modèle
2. Développer la configuration logicielle

IB- R3.04

12/69

## Classification des diagrammes UML



IB- R3.04

13/69

## Particularités des diagrammes de classes

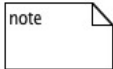
Cette partie complète votre connaissance des diagrammes de classes pour la phase de conception.

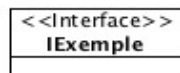
- Annotations des diagrammes UML
- Visibilité, direction et cardinalité
- Classes paramétrées
- Contraintes et sémantique des associations
- Modélisation des interfaces

IB- R3.04

14/69

## Annotations des diagrammes UML

- Note : pour insérer des commentaires libres dans les diagrammes  

- Propriété : à insérer à des endroits précis  
 {abstract} {leaf} {frozen} {author : ib}
- Contrainte  
 {ordered}
- Stéréotype : précise une catégorie et se met au dessus du nom de la classe ou interface  
 <<Exception>>, <<enumeration>>



IB- R3.04

15/69

## Diagramme de classes : contraintes sur les associations

Des contraintes peuvent être précisées sur les associations :

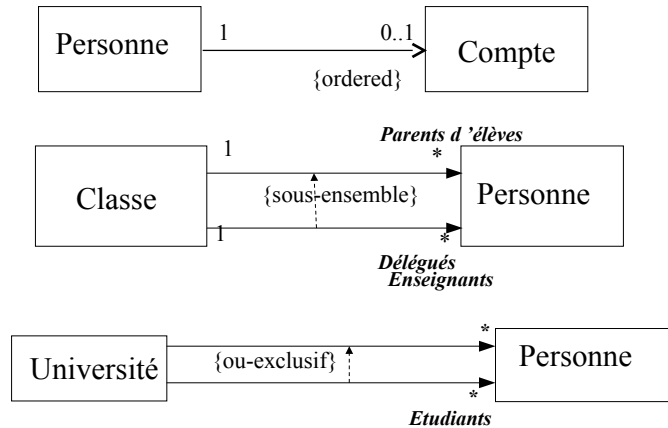
- ♦ ordre
- ♦ sous-ensemble (\*)
- ♦ ou-exclusif (\*)
- ♦ réflexion
- ♦ classes-association
- ♦ contraintes définie par l'utilisateur

(\*) généralement pas supporté par les ateliers UML

IB- R3.04

16/69

## Exemple de contraintes sur association



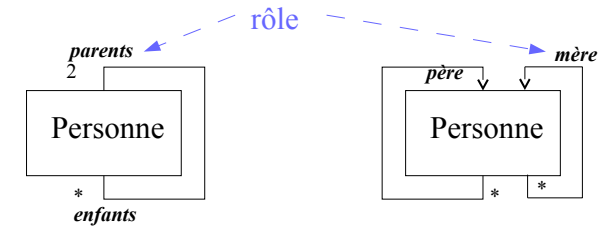
IB- R3.04

17/69

## Association réflexive

Les associations peuvent relier une classe à elle-même comme dans le cas de structures récursives.

Le **nommage des rôles** est très important.

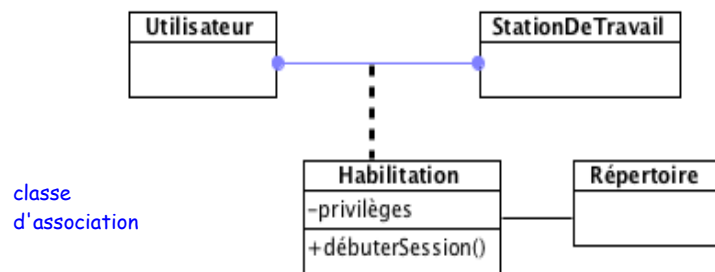


IB- R3.04

18/69

## Classe-Association (uniquement en phase d'analyse)

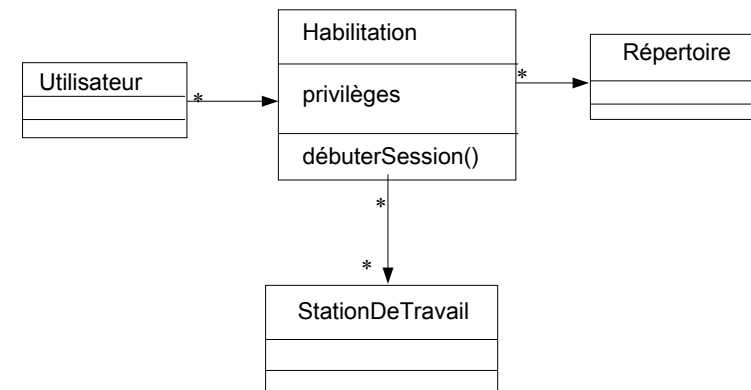
Une association peut être représentée par une classe pour ajouter des attributs et des opérations dans l'association.



IB- R3.04

19/69

## La classe d'association devient une classe à part entière (conception)



IB- R3.04

20/69

## Classes paramétrées (généricité)

- ♦ Une classe paramétrée décrit une classe avec des paramètres (de type généralement).
- ♦ Elle permet de définir une famille de classes par liaison des paramètres formels à des valeurs.
- ♦ La classe ne devient donc effective que lorsque cette liaison est faite.

IB- R3.04

21/69

## Classes paramétrées (templates)

Plusieurs langages, en particulier C++, possèdent la notion de classe paramétrée ou *template*.

Pour la généricité en Java, voir:

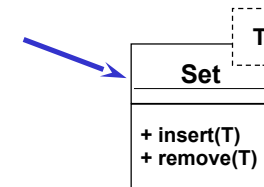
<https://docs.oracle.com/javase/tutorial/java/generics/types.html>

```
Class Set <T> {
    public void insert (T nouvelElement);
    public void remove (T unElement); }
```

Set<Employee> employeeSet

dérivation

Classe

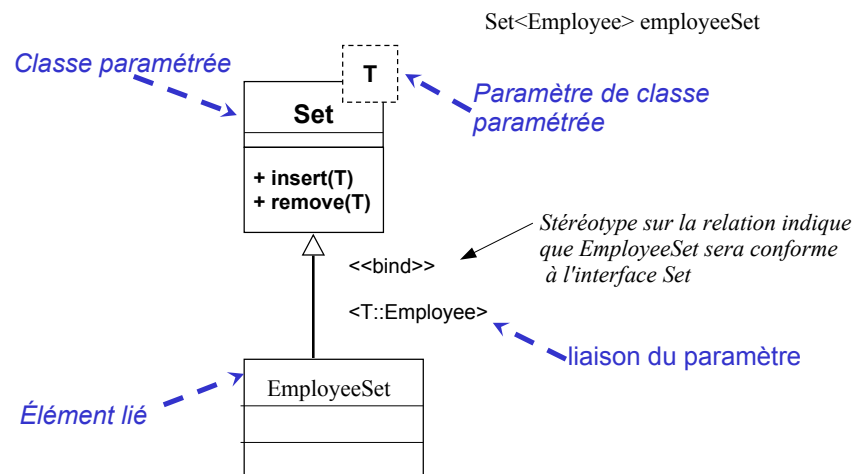


Paramètre de classe paramétrée

IB- R3.04

22/69

## Notation des classes paramétrées et dérivation



IB- R3.04

23/69

## Sémantique des associations

Agrégation

Composition

Généralisation et héritage

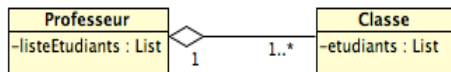
Classe abstraite

IB- R3.04

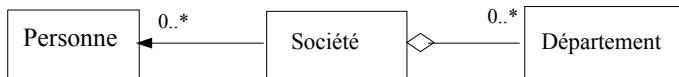
24/69

## Agrégation

- ♦ Forme particulière d'association entre un tout et ses composantes
- ♦ Association non symétrique.



*Ne pas confondre agrégation et association ordinaire.*

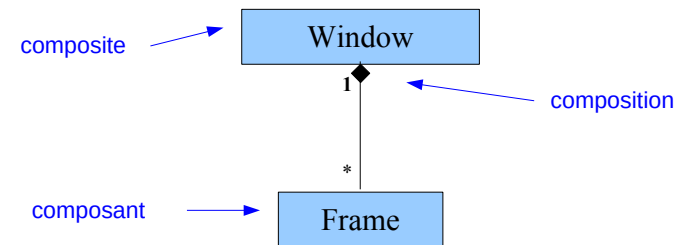


IB- R3.04

25/69

## Composition

- ♦ Une composition est une **forme plus forte** d'association
- ♦ Dans une composition l'objet composite doit gérer la création et la destruction de ses composants.



IB- R3.04

26/69

## Dépendance entre classes

— — — — — >

Une dépendance est une forme plus faible de lien qui indique qu'une classe dépend d'une autre à un moment donné.

Une classe dépend d'une autre si l'autre classe est utilisée pour typer :

- un paramètre d'une méthode « parameter »
- une variable locale dans une méthode « local »
- le retour d'une méthode « return »

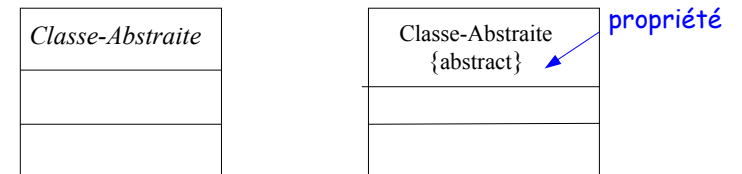
On pourra ajouter un stéréotype sur le lien de dépendance pour la préciser.

IB- R3.04

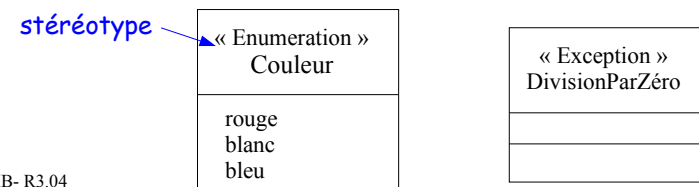
27/69

## Notations des classes particulières

Les noms des classes abstraites ne sont en italique que dans les ateliers



Les classes d'exception, énumération

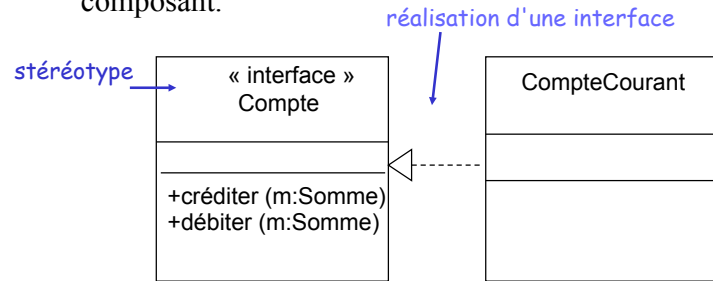


IB- R3.04

28/69

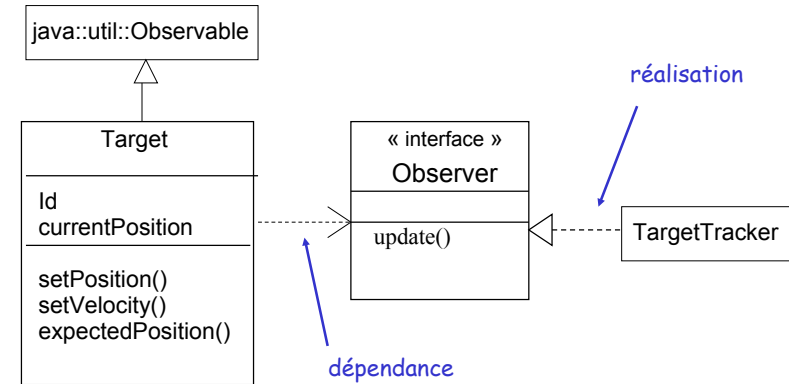
## Diagramme de classe de conception : Modélisation des interfaces

Une interface est un ensemble nommé de méthodes utilisées pour spécifier un service de classe ou de composant.

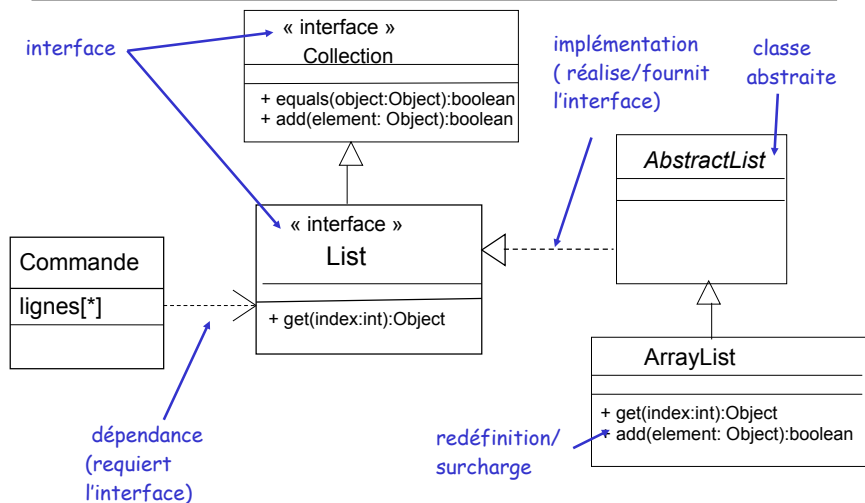


CompteCourant.implements Compte

## Réalisation/utilisation des interfaces



## Exemple d'interfaces et classe abstraite en Java



## Modéliser les interactions avec UML (modèle dynamique)



Interactions entre des objets d'un point de vue temporel :

- objet **émetteur** d'événements
- objet **récepteur** d'événements

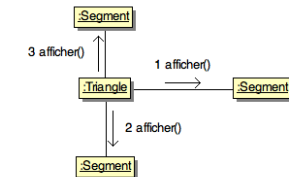
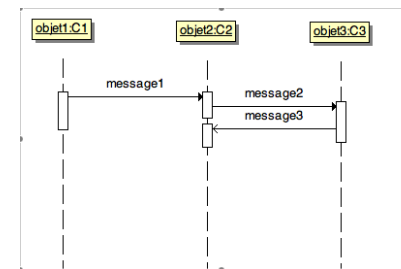
Communication (vue plus spatiale)

- des objets communiquent et collaborent pour accomplir une tâche donnée

## Les deux types de diagrammes d'interaction

diagramme de séquence

diagramme de communication



## Diagramme de séquence

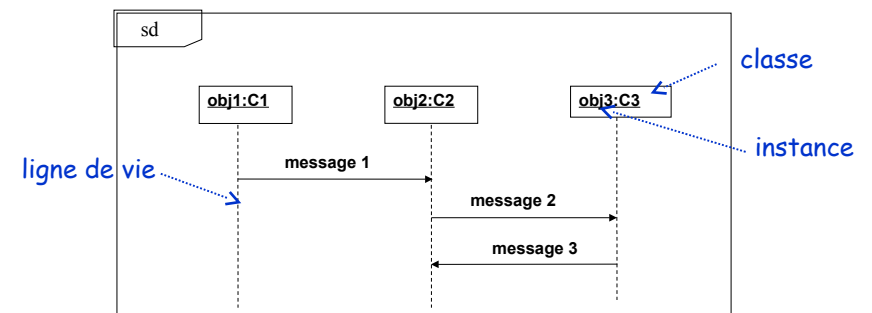
Il montre l'interaction des objets dans le temps et construit un flot séquentiel de messages.

Différentes utilisations selon la phase du cycle de vie et le niveau de détail désiré :

- **Documentation des cas d'utilisation** : description de l'interaction dans des termes proches de l'utilisateur et sans entrer dans les détails de synchronisation.
- **Usage dans la conception** : montrent les envois de messages échangés entre des objets pour réaliser une fonctionnalité.

## Diagramme de séquence : notation

La dimension **verticale** (modélise le temps qui passe )  
la dimension **horizontale** (pour les envois de messages )



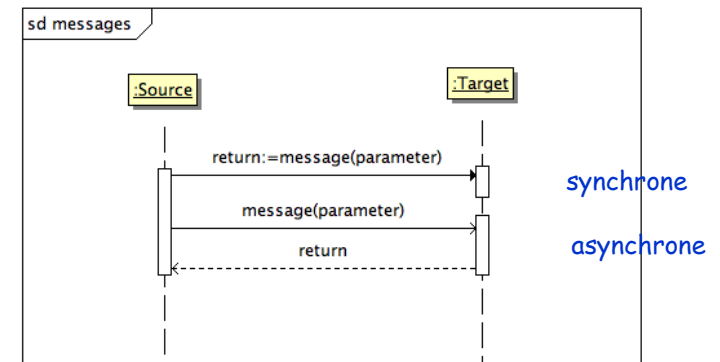
## Flots de contrôle des messages

Flot de **contrôle synchrone** : l'objet émetteur se bloque en attendant la réponse du récepteur du message.

Flot de **contrôle asynchrone** : l'objet émetteur n'attend pas la réponse du récepteur et poursuit sa tâche sans se soucier de la réception du message.

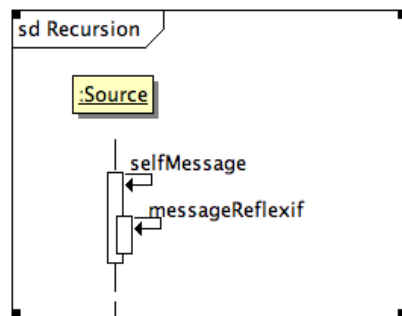
D'autres type de flots de contrôle : retour de procédure, timeout, etc. (notations graphiques spécifiques)

## Messages dans les diagrammes de séquence

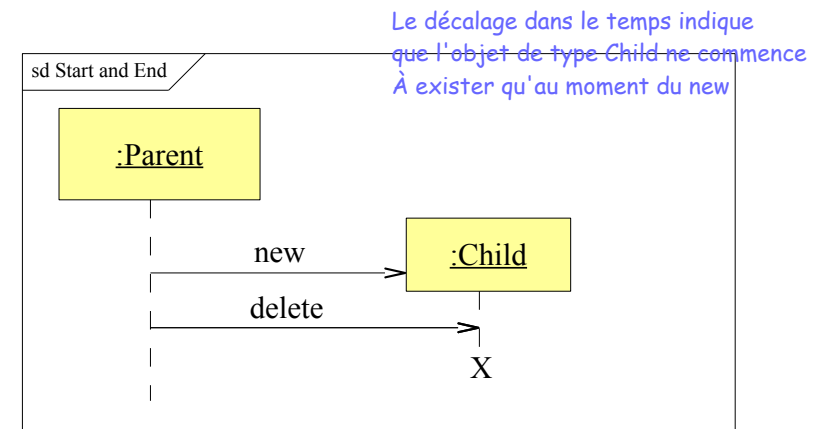


On ne prendra pas en compte ces nuances dans la terminaison de la flèche

## Diagramme de séquence : appel récursif d'une opération



## Diagramme de séquence : début et fin d'une ligne de vie



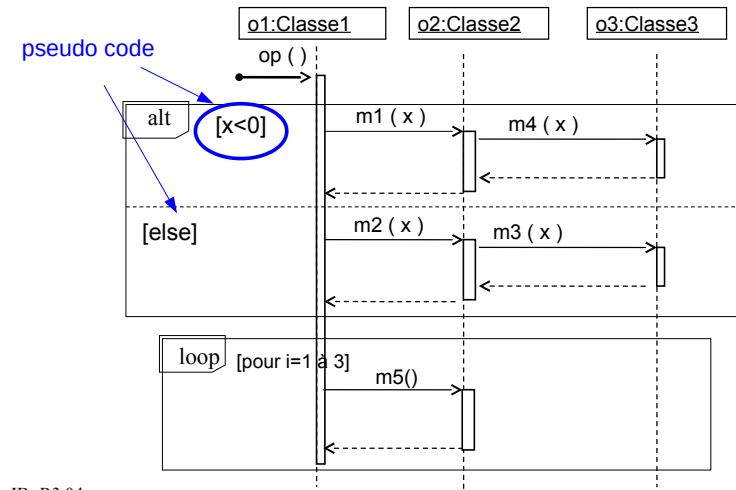
## Diagramme de séquence : opérations dans les cadres d'interaction

- **alt** : fragments multiple alternatifs (si alors sinon)
- **opt** : fragment optionnel
- **par** : fragment parallèle (traitements concurrents)
- **loop** : le fragment s'exécute plusieurs fois
- **region** : région critique (un seul thread à la fois)
- **neg** : une interaction non valable
- **ref** : référence à une interaction dans un autre diagramme
- **sd** : fragment du diagramme de séquence en entier

IB- R3.04

41/69

## Diagramme de séquence : Messages conditionnels et boucle



IB- R3.04

42/69

## Différences entre diagramme de séquence et diagramme de communication

Interactions entre des objets d'un point de vue temporel :

- chronologie explicite des envois de messages

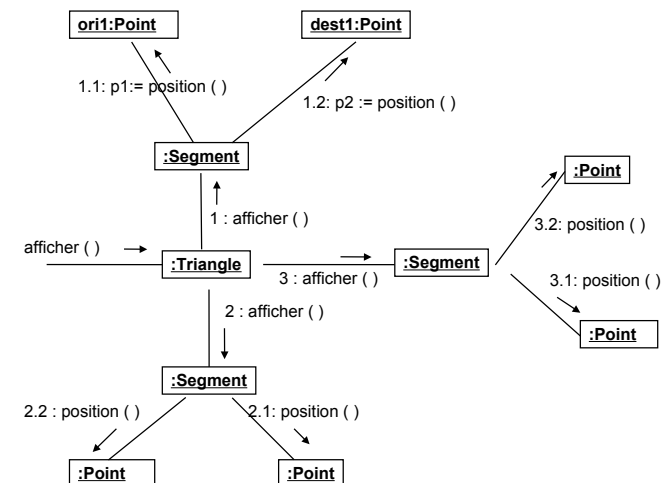
### Communication

- rend compte de l'organisation spatiale des participantes à une interaction
- on peut indiquer comment un objet est lié à un autre en attachant un stéréotype au chemin
- on indique l'ordre dans le temps des messages en utilisant une numérotation.

IB- R3.04

43/69

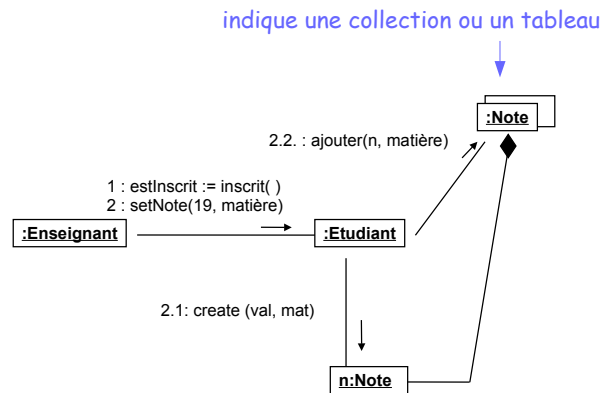
## Numérotation des messages pour indiquer l'ordre des envois de messages



IB- R3.04

44/69

## Diagramme de communication : multi-objet



IB- R3.04

45/69

## Scénarios génériques

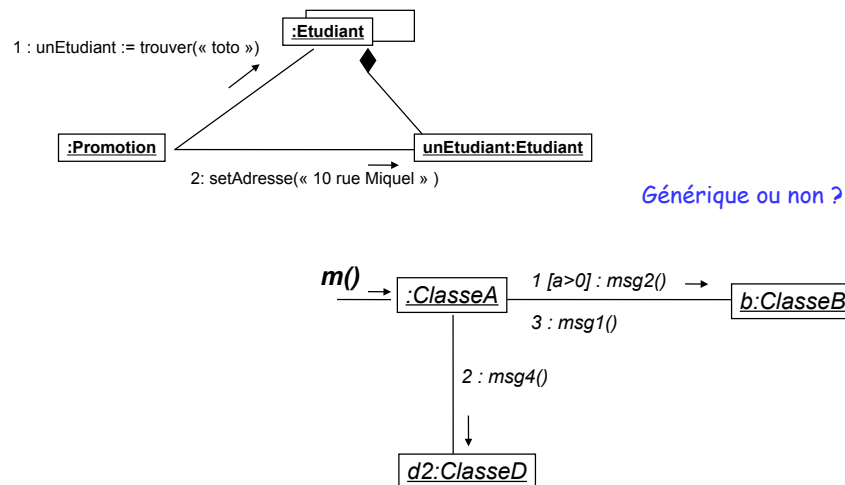
Une interaction sans branche conditionnelle décrit un scénario particulier.

Une interaction avec des branches conditionnelles décrit plusieurs situations possibles (algorithme) : scénario générique.

IB- R3.04

46/69

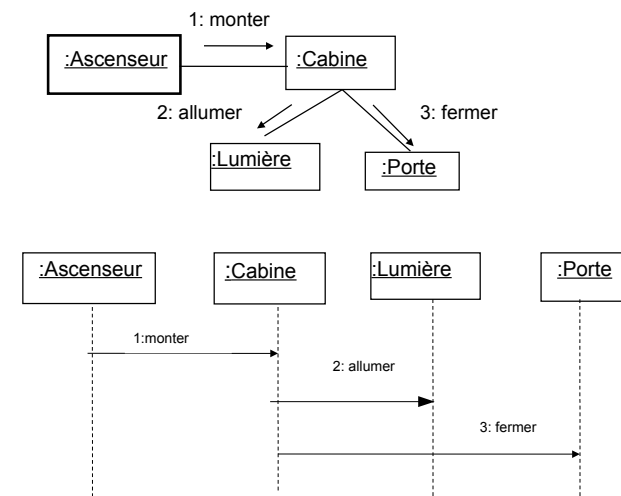
## Scénario particulier ou générique



IB- R3.04

47/69

## Équivalence entre diagramme de séquence et diagramme de communication



IB- R3.04

48/69

## Diagramme et niveaux de profondeur

Diagramme de niveau 1 : interactions entre l'objet considéré et ses fournisseurs (les objets dont il est le client).

Diagramme de niveau 2 : interactions entre l'objet considéré, ses fournisseurs et les fournisseurs de ses fournisseurs.

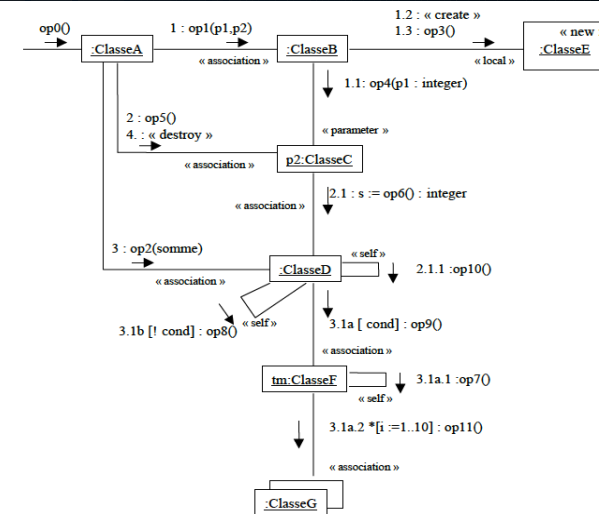
...

Diagramme de niveau N

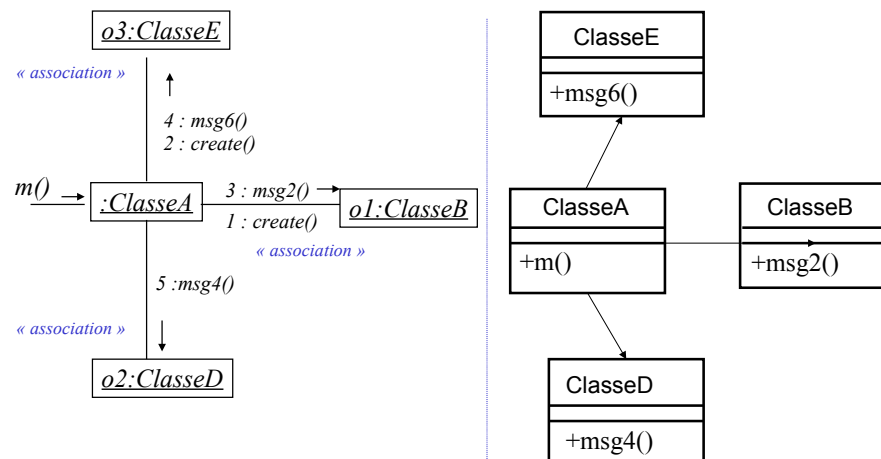
Un diagramme est complet s'il présente tous les objets de la modélisation.

## Séréotypes : diagramme de communication

(exemple qui sera vu en TD)



## Visibilité unidirectionnelle 1 - 1 :



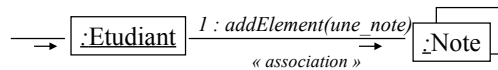
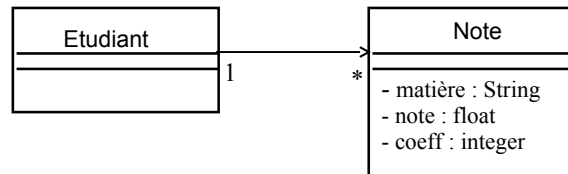
## Visibilité unidirectionnelle 1 - 1 : code Java

```

public class ClasseA {
    ...
    private ClasseB o1;
    private ClasseD o2;
    private ClasseE o3;
    ...
    public ClasseA (ClasseD par) {
        o2 = par; }
    public void m() {
        ...
        o1 = new ClasseB; ...;
        o3 = new ClasseE; ...;
        o1.msg2(); ...;
        o3.msg6(); ...;
        o2.msg4(); ...;
    }
}

```

## Visibilité unidirectionnelle 1 - N



Objet conteneur « **Notes** » (instance de la classe *ArrayList* ou *HashTable*) =



IB- R3.04

53/69

## Implémentation par défaut : visibilité de la structure

```

public class Note {
    private String matière;
    private float note;
    private int coeff;
    ...
}
  
```

```

import java.util.*;

public class Etudiant {

    private ArrayList<Notes> notes = new ArrayList<Note>();
    ...
    public void noter(Note uneNote) {
        notes.add(uneNote);
    }
}
  
```

IB- R3.04

54/69

## Implémentation : encapsulation totale (une classe pour la collection)

```

public class Note {
    // identique au cas précédent
}
  
```

```

public class Notes {
    // structure de données
}
  
```

```

public class Etudiant {
    private Notes notes;
    ...
}
  
```

IB- R3.04

55/69

## Détails de la classe **Notes**

```

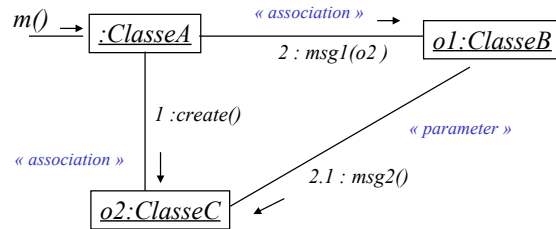
import java.util.*;
public class Notes {
    // surtout pas d'héritage avec Hashtable!
    private HashMap<String, Notes> lesNotes =
        new HashMap<String, Notes>();
    private float moyenne;

    public void ajouter(Note uneNote) {
        lesNotes.put(uneNote.getMatiere(), uneNote);
    }
    public Note chercher(String matiere) {
        return lesNotes.get(matiere);
    }
    public void afficher() {
        for(String matiere : lesNotes.keySet()) {
            System.out.println(matiere + " : " +
                lesNotes.get(matiere));
        }
    }
}
  
```

IB- R3.04

56/69

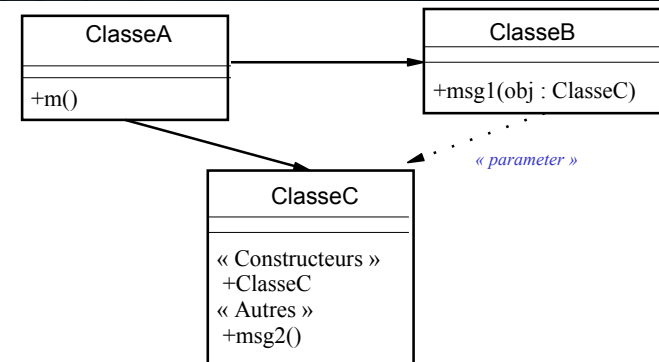
## Visibilité par paramètre (dépendance)



IB- R3.04

57/69

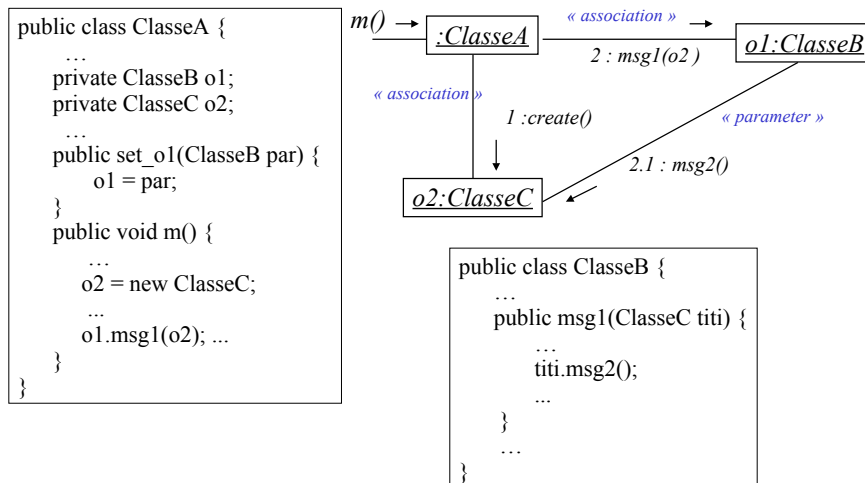
## Visibilité par paramètre



IB- R3.04

58/69

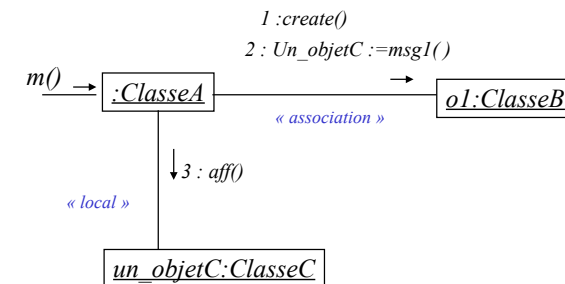
## Visibilité par paramètre : implémentation



IB- R3.04

59/69

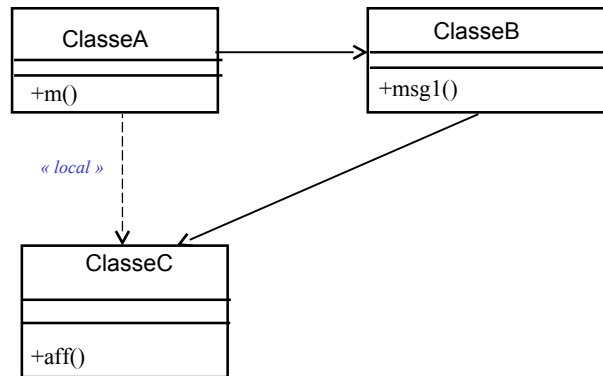
## Visibilité par variable locale d'une méthode



IB- R3.04

60/69

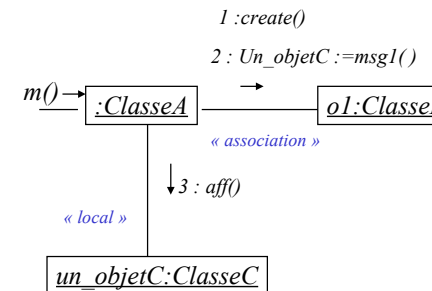
## Visibilité par variable d'une méthode



IB- R3.04

61/69

## Visibilité par variable d'une méthode générer le pseudo-code



```

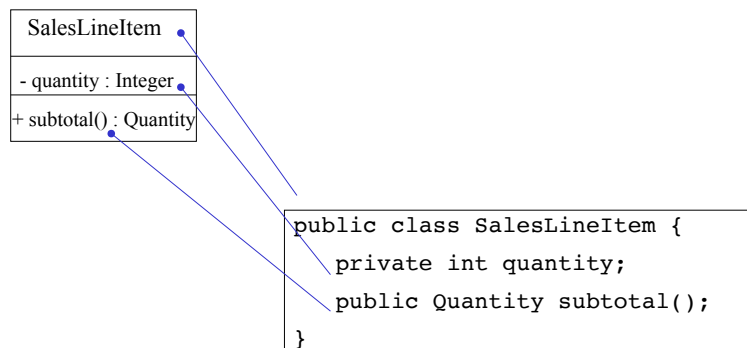
public class ClasseA {
    ...
    private ClasseB o1;
    ...
    public void m() {
        ClasseC un_objetC;
        ...
        o1 = new ClasseB;
        ...
        un_objetC = o1.msg1();
        ...
        un_objetC.aff();
        ...
    }
}
  
```

IB- R3.04

62/69

## Des classes de conception au code

Définition d'une classe avec des attributs simples et des méthodes

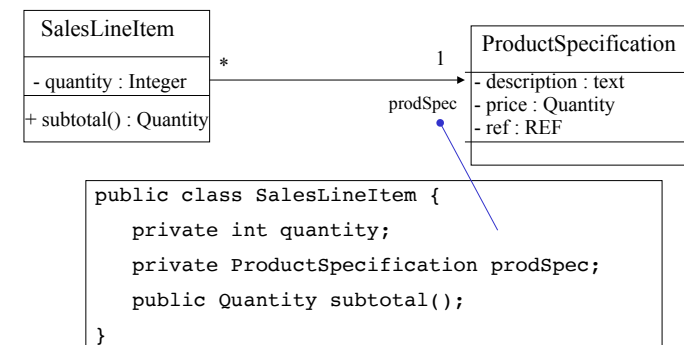


IB- R3.04

63/69

## Passer de la conception au code : variable d'instance

Ajout des attributs de référence et des noms de rôles utilisés pour les noms d'attributs.



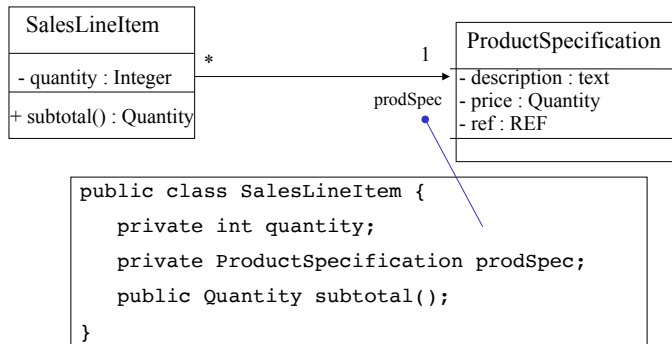
IB- R3.04

64/69



## Passer de la conception au code : variable d'instance

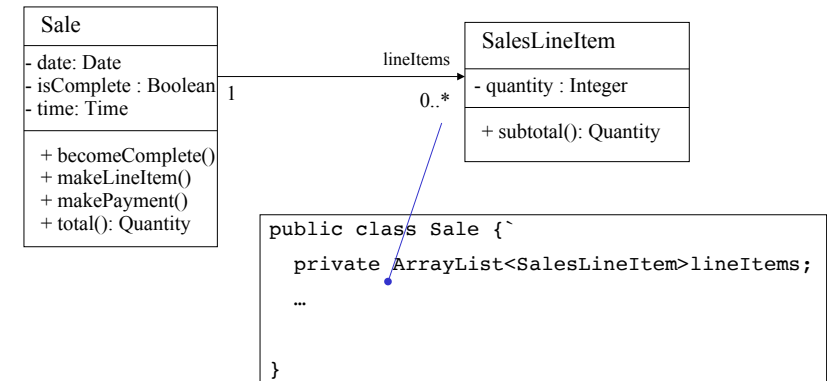
Ajout des attributs de référence et des noms de rôles utilisés pour les noms d'attributs.



IB- R3.04

65/69

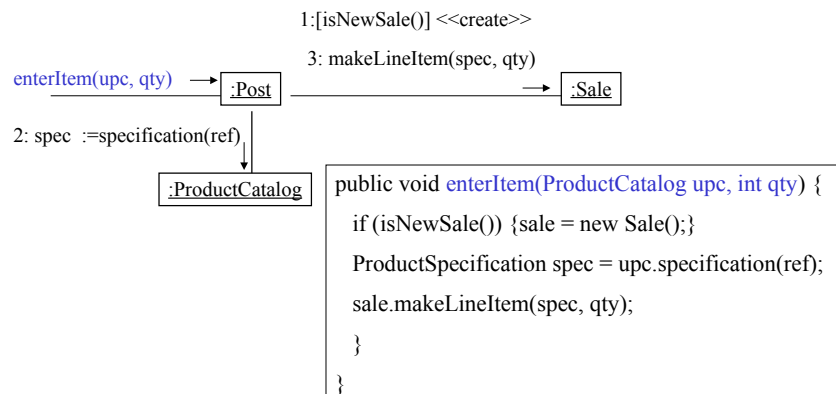
## Passer de la conception au code : association multiple



IB- R3.04

66/69

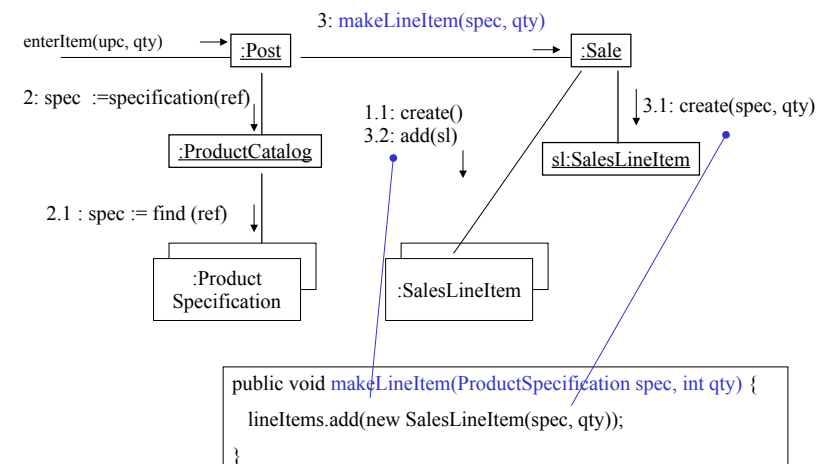
## Passer de la conception au code : code des méthodes



IB- R3.04

67/69

## Message avec un objet multiple



IB- R3.04

68/69

## Exercice : recherche et ajout d'un livre dans le panier

Ecrire le pseudo-code correspondant

