# Sémantique Statique : OCL

Xavier Crégut, Benoît Combemale, Arnaud Dieumegard, Marc Pantel

IRIT–ENSEEIHT
2, rue Charles Camichel - BP 7122
F-31071 Toulouse Cedex 7
{prenom.nom}@enseeiht.fr

Septembre 2019

# Sommaire

# Objectif général

**Objectif :** OCL est avant tout un **langage de requête** pour calculer une *expression sur un modèle en s'appuyant sur sa syntaxe* (son méta-modèle).
$\Longrightarrow$ Une expression exprimée une fois, pourra être évaluée sur tout modèle conforme au méta-modèle correspondant.

**Exemple :** pour une bibliothèque particulière on peut vouloir demander :

- Livres possédés par la bibliothèque ? Combien y en a-t-il ?
- Auteurs dont au moins un titre est possédé par la bibliothèque ?
- Titres dans la bibliothèque écrits par Martin Fowler ?
- Nombre de pages du plus petit ouvrage ?
- Nombre moyen de pages des ouvrages ?
- Ouvrages de plus 100 pages écrits par au moins trois auteurs ?
- ...

# Programmation par contrat

**Principe :** Établir formellement les responsabilités d'une classe et de ses méthodes.

**Moyen :** définition de propriétés (expressions booléennes) appelées :

- **invariant :** propriété définie sur une **classe** qui doit toujours être vraie, de la création à la disparition d'un objet.
  Un invariant lie les requêtes d'une classe (état externe).
- **précondition :** propriété sur une **méthode** qui :
  - doit être vérifiée par l'appelant pour que l'appel à cette méthode soit possible ;
  - peut donc être supposée vraie dans le code de la méthode.

  **postconditions :** propriété sur une **méthode** qui définit l'effet de la méthode, c'est-à-dire :
  - spécification de ce que doit écrire le programmeur de la méthode ;
  - caractérisation du résultat que l'appelant obtiendra.

**Exercice :** Invariant pour une Fraction (état = numérateur et dénominateur) ?

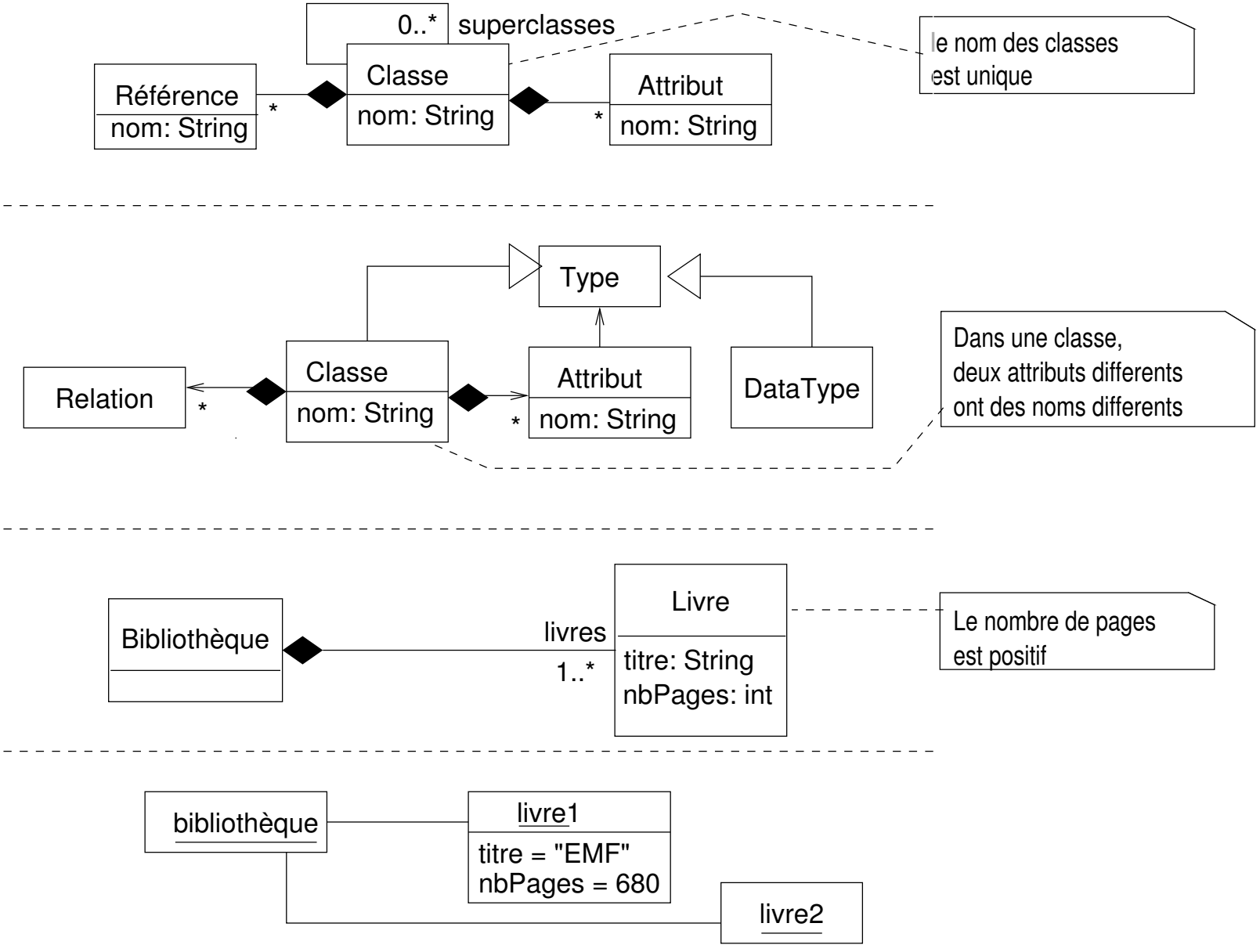
**Exercice :** Pré- et postconditions de racine carrée et de pgcd ?

# OCL et Diagrammes d'UML

OCL peut être utilisé sur différents diagrammes d'UML :

- diagramme de classe :
  - définir des préconditions, postconditions et invariants :
    Stéréotypes prédéfinis : «precondition», «postcondition» et «invariant»
  - caractérisation d'un **attribut dérivé** (p.ex. le salaire est fonction de l'âge)
  - spécifier la **valeur initiale** d'un attribut (p.ex. l'attribut *salaire* d'un employé)
  - spécifier le **code d'une opération** (p.ex. le salaire annuel est 12 fois le salaire mensuel)
- diagramme d'état :
  - spécifier une garde sur une transition
  - exprimer une expression dans une activité (affectation, etc.)
  - ...
- diagramme de séquence :
  - spécifier une garde sur un envoi de message
- ...

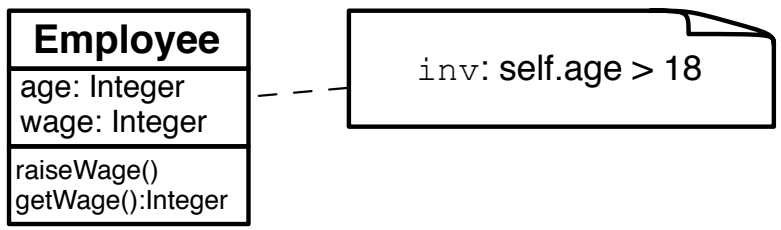
# OCL et Méta-modélisation : préciser la sémantique statique d'un modèle



# The *Object Constraint Language*

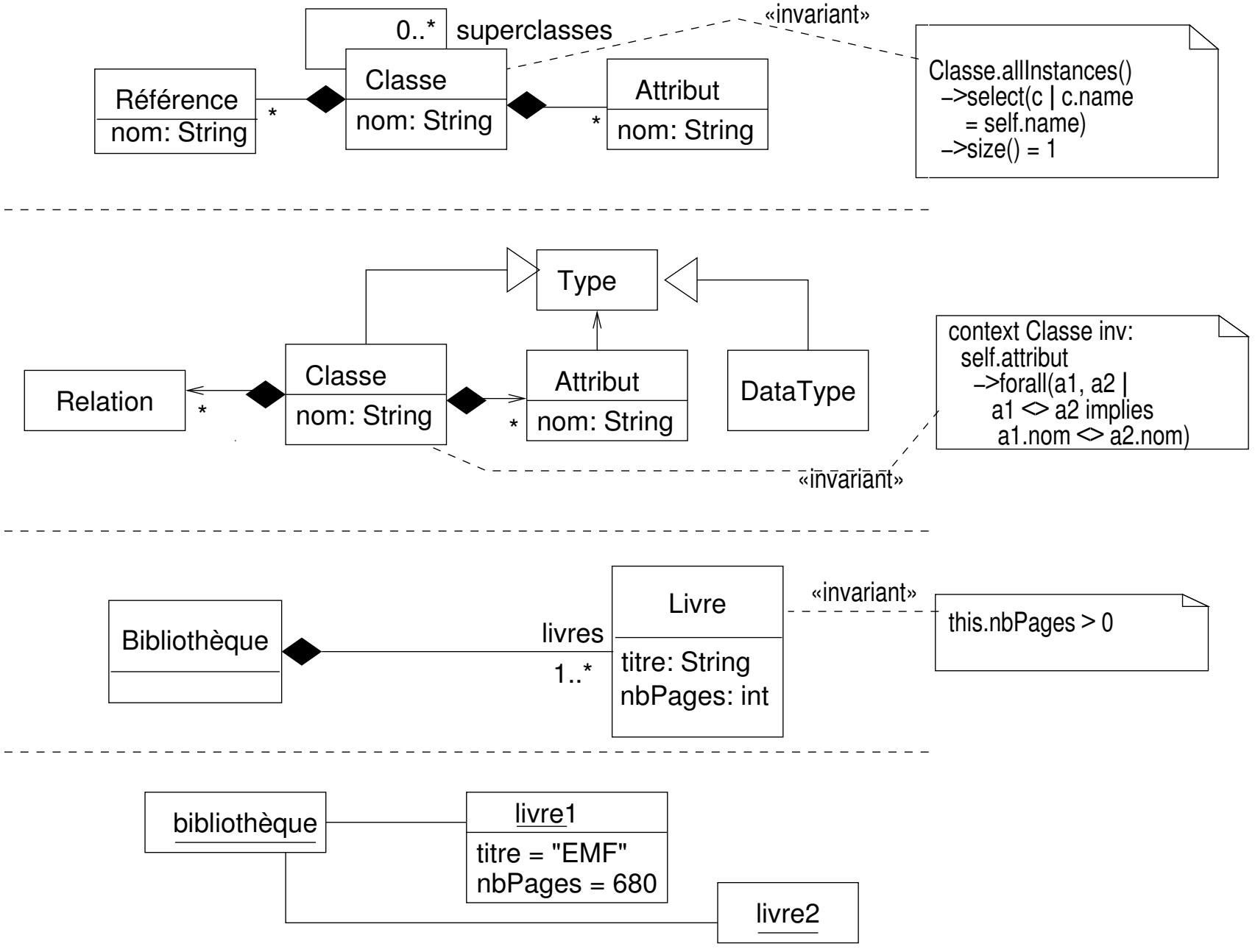
## Objectifs initiaux

- Les langages formels traditionnels (e.g. Z) requièrent de la part des utilisateurs une bonne compréhension des fondements mathématiques.
- *Object Constraint Language (OCL)* a été développé dans le but d'être :
  - formel, précis et non ambigu,




  - utilisable par un large nombre d'utilisateurs,
  - un langage de spécification (et non de programmation !),
  - supporté par des outils.
- Historique :
  - Développé en 1995 par IBM,
  - Inclu dans le standard UML jusqu'à la version 1.1 (1997),
  - OCL 2.0 *Final Adopted Specification* (ptc/06-05-01), May 2006.
  - Version actuelle : OCL 2.4 (formal/2014-02-03), January 2012.

# Préciser la sémantique statique d'un modèle



# The *Object Constraint Language*

## Propriétés du langage

- **Langage de spécification sans effet de bord**
  - une expression OCL calcule une valeur... **et** laisse le modèle inchangé !
    ⇒ l'état d'un objet ne peut pas être modifié **par** l'évaluation d'une expression OCL
  - l'évaluation d'une expression OCL est instantanée
    ⇒ l'état des objets ne peut donc pas être modifié **pendant** l'évaluation d'une expression OCL
  - OCL n'est pas un langage de programmation !
- OCL est un **langage typé** :
  - Chaque expression OCL a un type
  - OCL définit des types primitifs : **Boolean**, **Integer**, **Real** et **String**
  - Chaque *Classifier* du modèle est un nouveau type OCL
  - *Intérêt :* vérifier la cohérence des expressions
    exemple : il est interdit de comparer un String et un Integer

# Les types OCL de base

Les types de base (*Primitive*) sont **Integer**, **Real**, **Boolean** et **String**. Les opérateurs suivants s'appliquent sur ces types :

| | |
|---|---|
| Opérateurs relationnels | $=, <>, >, <, >=, <=$ |
| Opérateurs logiques | *and*, *or*, *xor*, *not*, *if* . . . *then* . . . *else* . . . *endif* |
| Opérateurs mathématiques | $+, -, /, *, min(), max()$. . . |
| Opérateurs pour les chaînes de caractères | *concat*, *toUpper*, *substring*... |

**Attention :** Concernant l'opérateur **if** ... **then** ... **else** ... **endif** :

- la clause **else** est nécessaire et,
- les expressions du **then** et du **else** doivent être de même type.

**Attention : and**, **or**, **xor** ne sont pas évalués en court-circuit !

# Priorité des opérateurs

Liste des opérateurs dans l'ordre de priorité décroissante :

```
1   @pre
2   .    ->                  -- notation pointée et fléchée
3    not  -                   -- opérateurs unaires
4   *    /
5   +    -                   -- opérateurs binaires
6    if- then- else- endif
7   <    >    <=    >=
8   =    <>
9    and   or    xor
10  implies                  -- implication
```

**Remarque :** Les parenthèses peuvent être utilisées pour changer la priorité.

# Les autres types OCL

- Tous les éléments du modèle sont des types (*OclModelElementType*),
  - y compris les énumérations : *Gender* :: *male*,
- Type *Tuple* : enregistrement (produit cartésien de plusieurs types)

  $$Tuple\ \{a : Collection(Integer) = Set\{1, 3, 4\}, b : String = 'foo'\}$$

- *OclMessageType* :
  - utilisé pour accéder aux messages d'une opération ou d'un signal,
  - offre un rapport sur la possibilité d'envoyer/recevoir une opération/un signal.
- *VoidType* :
  - a seulement une instance *oclUndefined*,
  - est conforme à tous les types.

# Contexte d'une expression OCL

Une expression est définie sur un **contexte** qui identifie :

- une **cible** : l'élément du modèle sur lequel porte l'expression OCL

| | | |
|---|---|---|
| T | Type (Classifier : Interface, Classe...) | **context** Employee |
| M | Opération/Méthode | **context** Employee::raiseWage(inc:Int) |
| A | Attribut ou extrémité d'association | **context** Employee::job : Job |

- le **rôle** : indique la **signification** de cette expression (pré, post, invariant...) et donc contraint sa **cible** et son **évaluation**.

| **rôle** | **cible** | **signification** | **évaluation** |
|---|---|---|---|
| **inv** | T | invariant | toujours vraie |
| **pre** | M | précondition | avant tout appel de M |
| **post** | M | postcondition | après tout appel de M |
| **body** | M | résultat d'une requête | appel de M |
| **init** | A | valeur initiale de A | création |
| **derive** | A | valeur de A | utilisation de A |
| **def** | T | définir une méthode ou un attribut | |

# Syntaxe d'OCL

**inv** (invariant) doit toujours être vrai (avant et après chaque appel de méthode)

**context** Employee
**inv**: self.age > 18

**context** e : Employee
**inv** age_18: e.age > 18

**pre** (precondition) doit être vraie avant l'exécution d'une opération

**post** (postcondition) doit être vraie après l'exécution d'une opération

**context** Employee::raiseWage(increment : **Integer**)
**pre**: increment > 0
**post** my_post: self.wage = self.wage**@pre** + increment

**context** Employee::getWage() : **Integer**
**post**: result = self.wage

**Remarques :** result et **@pre** : utilisables seulement dans une postcondition

- **exp@pre** correspond à la valeur de expr avant l'appel de l'opération.
- result est une variable prédéfinie qui désigne le résultat de l'opération.

# Syntaxe d'OCL

- **body** spécifie le résultat d'une opération

  **context** Employee::getWage() : **Integer**
  **body**: self.wage

- **init** spécifie la valeur initiale d'un attribut ou d'une association

  **context** Employee::wage : **Integer**
  **init**: 900

- **derive** spécifie la règle de dérivation d'un attribut ou d'une association

  **context** Employee::wage : **Integer**
  **derive**: self.age $*$ 50

- **def** définition d'opérations (ou variables) qui pourront être (ré)utilisées dans des expressions OCL.

  **context** Employee
  **def**: annualIncome : **Integer** = 12 $*$ wage

# La navigation dans le modèle

## Accès aux informations de la classe

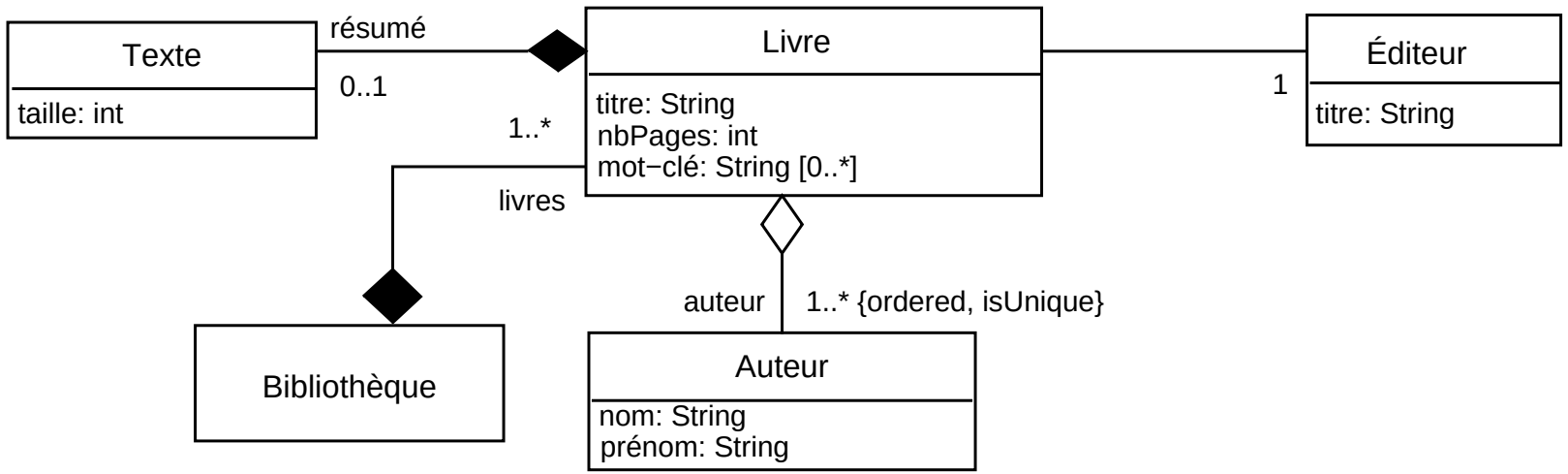
- Une expression OCL est définie dans le contexte d'une classe
  - en fait : un type, une interface, une classe, etc.
- Elle s'applique sur un objet, instance de cette classe :

  $\Longrightarrow$ cet objet est désigné par le mot-clé **self**.
- Étant donné un accès à un objet (p.ex. self), une expression OCL peut :
  - accéder à la valeur des attributs :
    - self.nbPages
    - unLivre.nbPages
  - appeler toute requête définie sur l'objet :
    - self.getNbPages()
    - unLivre.getNbPages()

    *Rappel :* Une requête (notée *{isQuery}* en UML) est une opération :
    - qui a un type de retour (calcule une expression) ;
    - et n'a pas d'effet de bord (ne modifie pas l'état du système).
  - parcourir les associations...

# Correspondance entre association et OCL

- Exemple de diagramme de classe




- pour atteindre l'autre extrémité d'une association, on utilise :
  - le rôle, p.ex. : unLivre.résumé
  - à défaut le nom de la classe en minuscule : unLivre.éditeur
- La manière dont une association est vue en OCL dépend :
  - de sa multiplicité : un exactement (1), optionnel (0..1), $\geq 2$
  - de ses qualificatifs : { **isUnique**}, { **isOrdered**}

# Correspondance entre association et OCL

## association avec multiplicité $\leq 1$

- multiplicité 1 : nécessairement un objet à l'extrémité (invariant implicite)
  - unLivre.éditeur
- multiplicité 0..1 (optionnel) :
  - utiliser l'opération **oclIsUndefined**()
  - unLivre.résumé. **oclIsUndefined**() est :
    - vraie si pas de résumé,
    - faux sinon
  - Exemple d'utilisation :

```
if unLivre.résumé.oclIsUndefined() then
    true
else
    unLivre.résumé.taille >= 60
endif
```

# Correspondance entre association et OCL

## association avec multiplicité $\geq 2$

- les éléments à l'extrémité d'une association sont accessibles par une collection
- OCL définit quatre types de collection :
  - **Set** : pas de double, pas d'ordre
  - **Bag** : doubles possibles, pas d'ordre
  - **OrderedSet** : pas de double, ordre
  - **Sequence** : doubles possibles, ordre
- Lien entre associations UML et collections OCL

| UML | Ecore | OCL |
|---|---|---|
| | | Bag |
| isUnique | Unique | Set |
| isOrdered | Ordered | Sequence |
| isUnique, isOrdered | Unique, Ordered | OrderedSet |

- Exemple : unLivre.auteur : la collection des auteurs de unLivre

# Les collections OCL

- *Set* : ensemble d'éléments *sans* doublon et *sans* ordre

  **Set** {7, 54, 22, 98, 9, 54, 20..25}
  —— *Set{7,54,22,98,9,20,21,23,24,25} : Set(Integer)*
  —— *ou Set{7,9,20,21,22,23,24,25,54,98} : Set(Integer), ou...*

- *OrderedSet* : ensemble d'éléments *sans* doublon et *avec* ordre

  **OrderedSet** {7, 54, 22, 98, 9, 54, 20..25}
  —— *OrderedSet{7,9,20,21,22,23,24,25,54,98} : OrderedSet(Integer)*

- *Bag* : ensemble d'éléments *avec* doublons possibles et *sans* ordre

  **Bag** {7, 54, 22, 98, 9, 54, 20..25}
  —— *p.ex. : Bag{7,9,20,21,22,22,23,24,25,54,54,98} : Bag(Integer)*

- *Sequence* : ensemble d'éléments *avec* doublons possibles et *avec* ordre

  **Sequence**{7, 54, 22, 98, 9, 54, 20..25}
  —— *Sequence{7,54,22,98,9,54,20,21,22,23,24,25} : Sequence(Integer)*

Les collections sont génériques : Bag(Integer), Set(String), Bag(Set(Livre))

# Opérations sur les collections (bibliothèque standard)

## Pour tous les types de Collection

**size**(): **Integer** —— *nombre d'éléments dans la collection self*
**includes**(object: T): **Boolean** —— *est−ce que object est dans sefl ?*
**excludes**(object: T): **Boolean** —— *est−ce que object n'est pas dans self ?*
**count**(object: T): **Integer** —— *nombre d'occurrences de object dans self*
**includesAll**(c2: **Collection**(T)): **Boolean**
—— *est−ce que self contient tous les éléments de c2 ?*
**excludesAll**(c2: **Collection**(T)): **Boolean**
—— *est−ce que self ne contient aucun des éléments de c2 ?*
**isEmpty**(): **Boolean** —— *est−ce que self est vide ?*
**notEmpty**(): **Boolean** —— *est−ce que self est non vide ?*
**sum**(): T —— *la somme (+) des éléments de self*
—— *l'opérateur + doit être défini sur le type des éléments de self*
**product**(c2: **Collection**(T2)): **Set**( **Tuple**(premier: T, second: T2))
—— *le produit (∗) des éléments de self*

# Opérations de la bibliothèque standard pour les collections

En fonction du sous-type de *Collection*, d'autres opérations sont disponibles :

- union
- intersection
- append
- flatten
- =
- ...

Une liste exhaustive des opérations de la bibliothèque standard pour les collections est disponible dans [OMG OCL 2.3.1, §11.7].

# Opérations de la bibliothèque standard pour tous les objets

OCL définit des opérations qui peuvent être appliquées à tous les objets

- *oclIsTypeOf*(*t* : *OclType*) : *Boolean*
  Le résultat est vrai si le type de *self* et *t* sont identiques.

  **context** Employee
  **inv**: self. **oclIsTypeOf**(Employee) −− *is true*
  **inv**: self. **oclIsTypeOf**(Company) −− *is false*

- *oclIsKindOf*(*t* : *OclType*) : *Boolean*
  vrai si *t* est le type de *self* ou un super-type de de *self*.
- *oclIsNew*() : *Boolean*
  Uniquement dans les post-conditions
  vrai si le récepteur a été créé au cours de l'exécution de l'opération.
- *oclIsInState*(*t* : *OclState*) : *Boolean*
  Le résultat est vrai si l'objet est dans l'état *t*.

# Opérations de la bibliothèque standard pour tous les objets

OCL définit des opérations qui peuvent être appliquées à tous les objets

- $oclAsType(t : OclType) : T$
  Retourne le même objet mais du type $t$
  Nécessite que $oclIsKindOf(t) = true$
- $allInstances()$
  - prédéfinie pour les classes, les interfaces et les énumérations,
  - le résultat est la collection de toutes les instances du type au moment de l'évaluation.

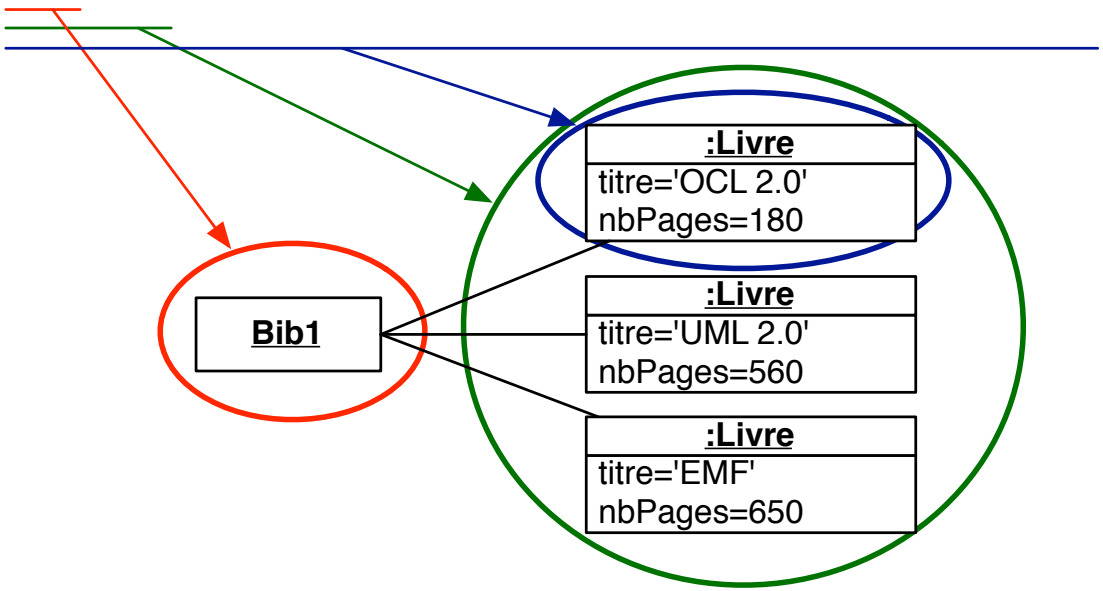
```
context Employee
    inv: Employee. allInstances()−> forAll(p1, p2
        | p1 <> p2 implies p1.name <> p2.name)
```

# Opérateur *select* (resp. *reject*)

Permet de spécifier le sous-ensemble de tous les éléments de *collection* pour lesquels l'expression est vraie (resp. fausse pour *reject*).

- $collection \rightarrow select(elem : T|expr)$
- $collection \rightarrow select(elem|expr)$
- $collection \rightarrow select(expr)$

**context Bibliothèque inv:**
**self.livres->select(name = 'OCL 2.0')->notEmpty()**

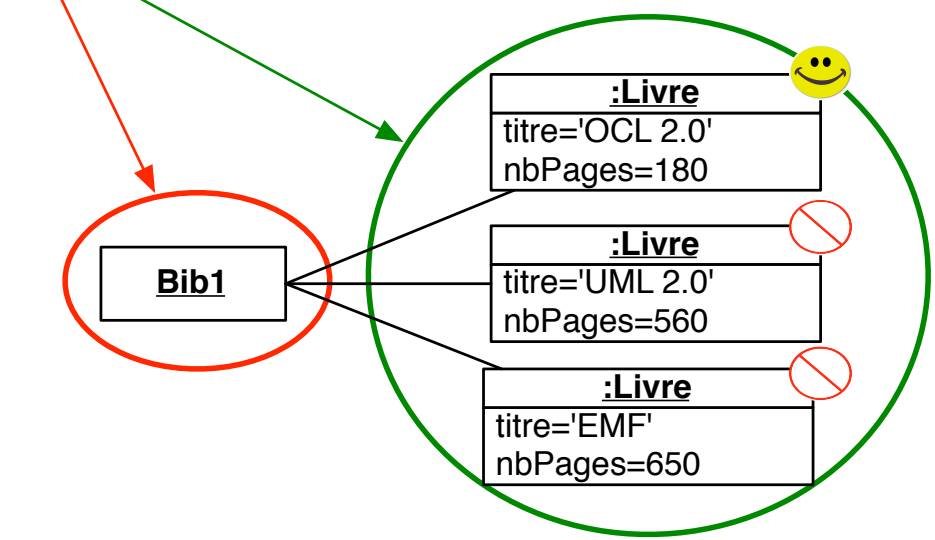


# Opérateur *exists*

Retourne vrai si l'expression est vraie pour au moins un élément de la collection.

- $collection \rightarrow exists(elem : T|expr)$
- $collection \rightarrow exists(elem|expr)$
- $collection \rightarrow exists(expr)$

**context Bibliothèque inv:**
**self.livres->exists(name = 'OCL 2.0')**

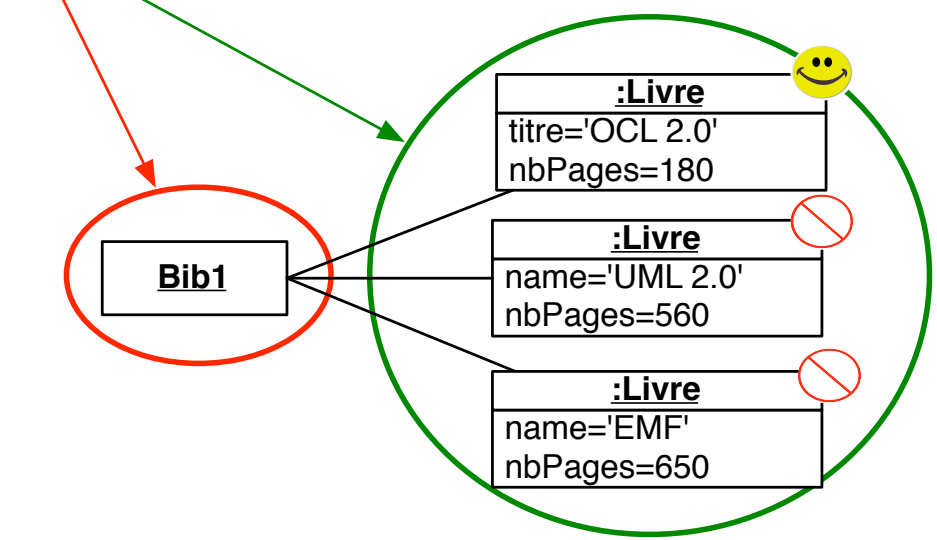


# Opérateur *forAll*

Retourne vrai si l'expression est vraie pour tous les éléments de la collection.

- $collection \rightarrow forAll(elem : T|expr)$
- $collection \rightarrow forAll(elem|expr)$
- $collection \rightarrow forAll(expr)$

**context Bibliothèque inv:**
**self.livres->forAll(nbPages < 200)**

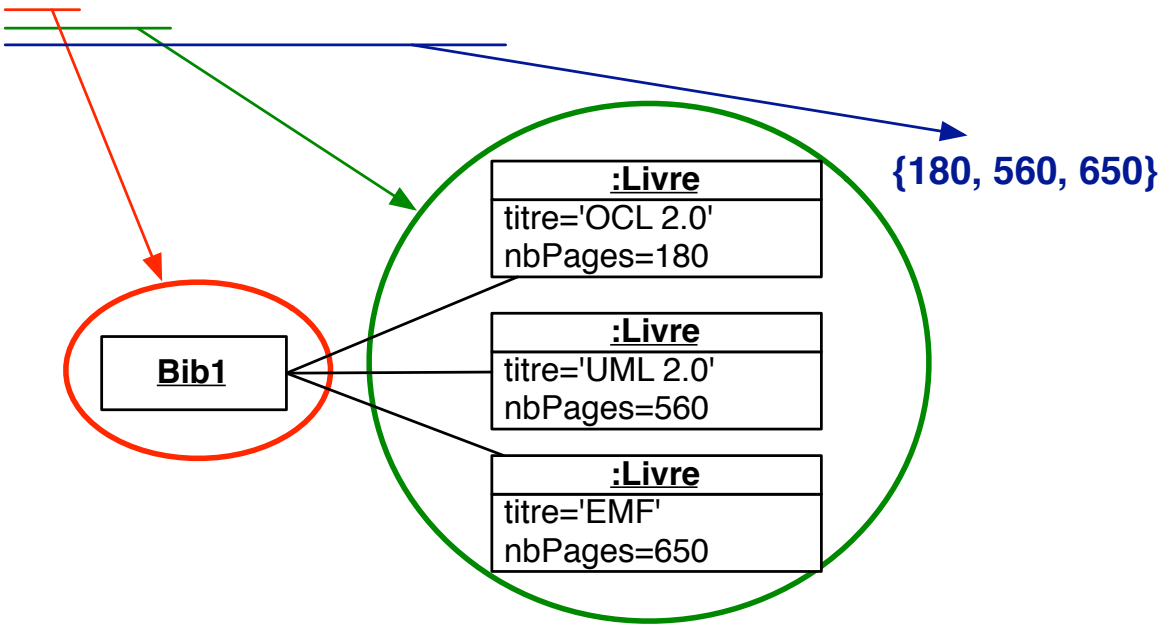


# Opérateur *collect*

Retourne la collection des valeurs (*Bag*) résultant de l'évaluation de l'expression appliquée à tous les éléments de collection.

- $collection \rightarrow collect(elem : T|expr)$
- $collection \rightarrow collect(elem|expr)$
- $collection \rightarrow collect(expr)$

**context Bibliothèque def moyenneDesPages : Real =**
**self.livres->collect(nbPages)->sum() / self.livres->size()**



# Opérateur *iterate*

Forme générale d'une itération sur une collection et permet de redéfinir les précédents opérateurs.

```
collection−> iterate(elem : Type;
        answer : Type = <value>
        | <expression_with_elem_and_answer>)
```

```
context Bibliothèque def moyenneDesPages : Real =
    self.livres−> collect(nbPages)−> sum() / self.livres−> size()

−− est identique à :

context Bibliothèque def moyenneDesPages : Real =
    self.livres−> iterate(l : Livre;
            lesPages : Bag{Integer} = Bag{}
            | lesPages−> including(l.nbPages))
        −> sum() / self.livres−> size()
```

# Plusieurs itérateurs pour un même opérateur

**Remarque :** les opérateurs *forAll*, *exist* et *iterate* acceptent plusieurs itérateurs :

```
Auteur. allInstances()−> forAll(a1, a2 |
        a1 <> a2 implies
            a1.nom <> a2.nom  or a1.prénom <> a2.prénom)
```

Bien sûr, dans ce cas il faut nommer tous les itérateurs !

# Conseils

**OCL ne remplace pas les explications en langage naturel.**

- Les deux sont *complémentaires* !
- *Comprendre* (informel)
- *Lever les ambiguités* (OCL)

**Éviter les expressions OCL trop compliquées**

- éviter les navigations complexes (utiliser **let** ou **def**)
- bien choisir le contexte (associer l'invariant au bon type !)
- éviter d'utiliser **allInstances**() :
  - rend souvent les invariants plus complexes
  - souvent difficile d'obtenir toutes les instances dans un système (sauf BD !)
- dissocier une conjonction de containtes en plusieurs (inv, post, pre)
- Toujours nommer les extrémités des associations (rôle des objets)