



• Analyse Orientée Objet

Cours 3 : Object Constraint Language (OCL)

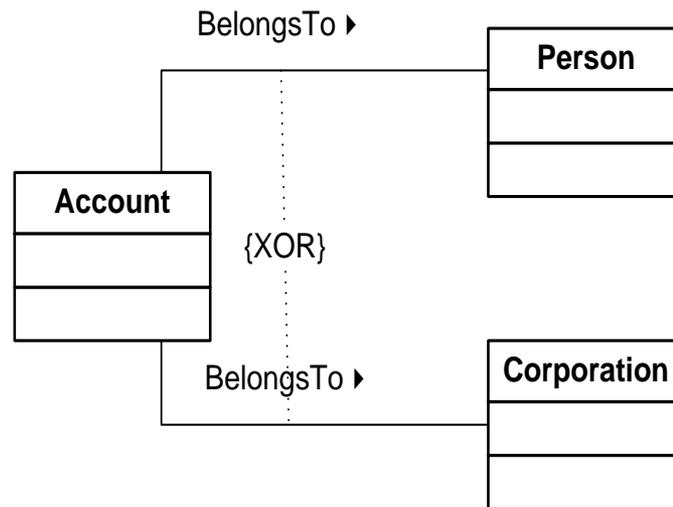
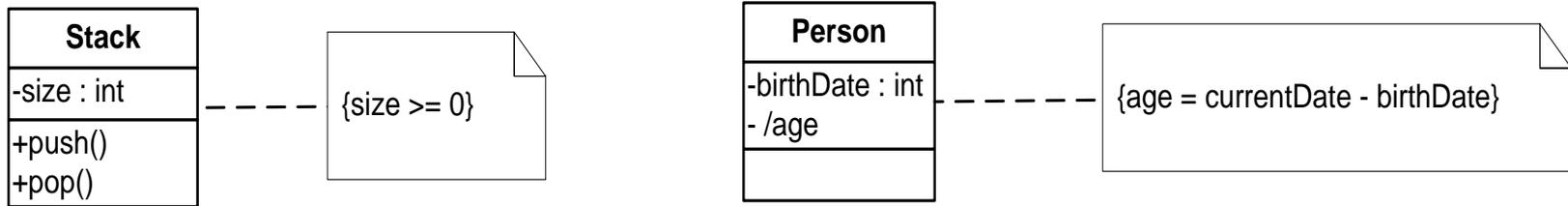
C'est quoi OCL?

- Un langage pour exprimer les contraintes dans les modèles UML.
- Un langage formel (sans ambiguïté) qui peut être lu et compris par les différents développeurs et clients
- Langage déclaratif : il décrit le *quoi* mais pas le *comment*
- Dernière version : OCL 2.2

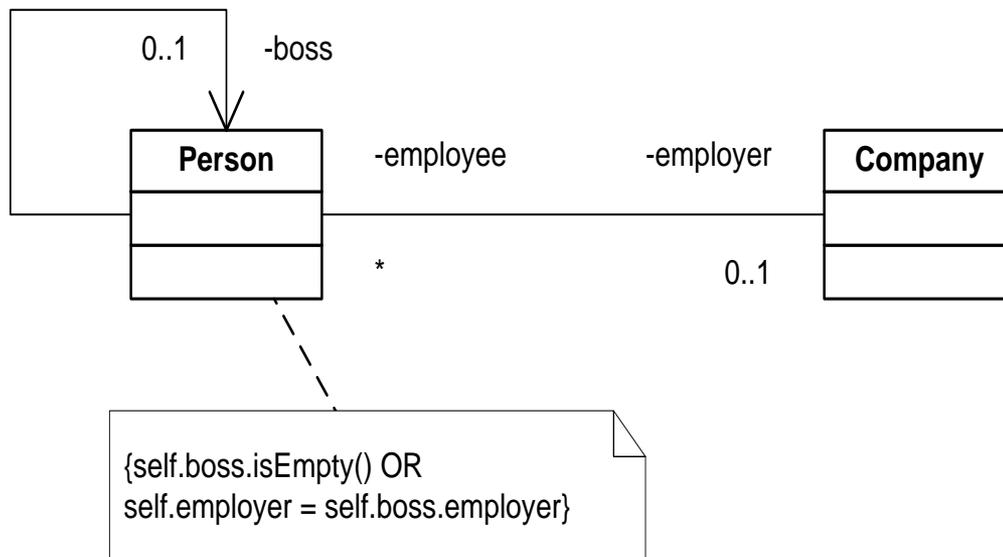
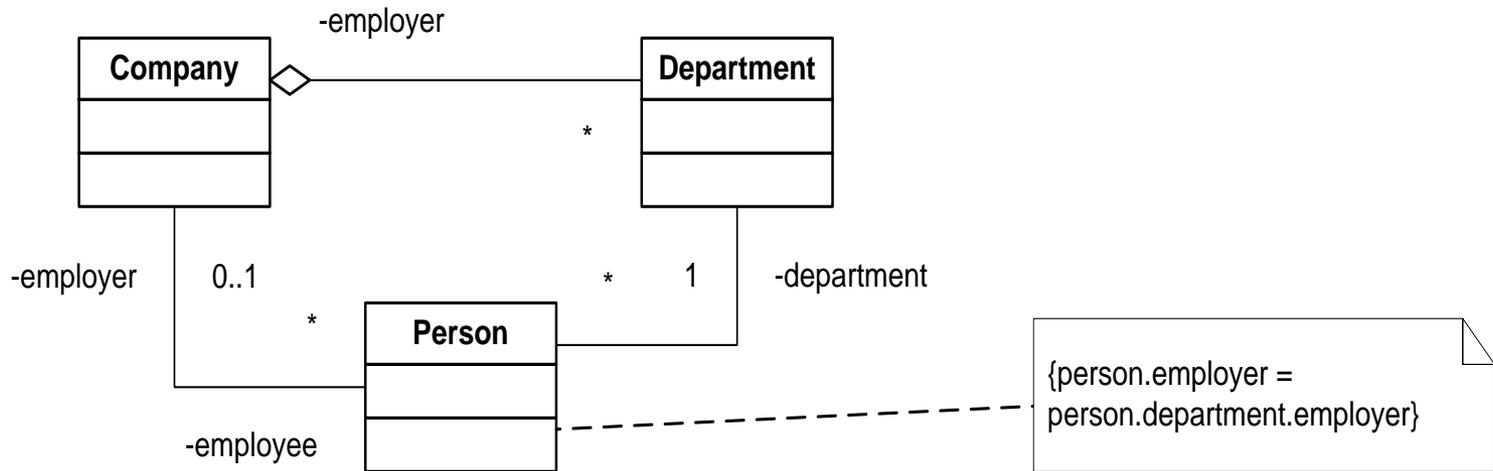
C'est quoi une contrainte OCL ?

- Une expression booléenne : *true* ou *false*
- Dans un note/texte/éditeur de contraintes dans un diagramme UML
 - Syntaxe : {contrainte}
- Trois types de contraintes :
 - Invariants
 - Pré-conditions
 - Post-conditions

Contrainte OCL : exemples



Contrainte OCL : encore des exemples



Contrainte OCL : contexte

- Une expression OCL est toujours associée à un élément de modèle (classe, attribut, relation, ...) : c'est le contexte de la contrainte.
- Il **existe deux manières** pour spécifier le contexte d'une contrainte OCL :
 - 1) En écrivant la contrainte entre accolades dans une note. L'élément pointé par la note est alors le contexte de la contrainte.
 - 2) En utilisant le mot-clef **context** dans un document accompagnant le diagramme.

OCL : contexte

- Syntaxe : **context** élément
- Élément peut être une classe, un attribut, une opération,
- Pour faire référence à un élément d'une classe, il faut utiliser les **::** comme séparateur
- **Exemples**
 - Le contexte est la classe *Compte*:
context Compte
 - Le contexte est l'opération *getSolde()* de la classe *Compte*:
context Compte::getSolde()

Contraintes : invariants

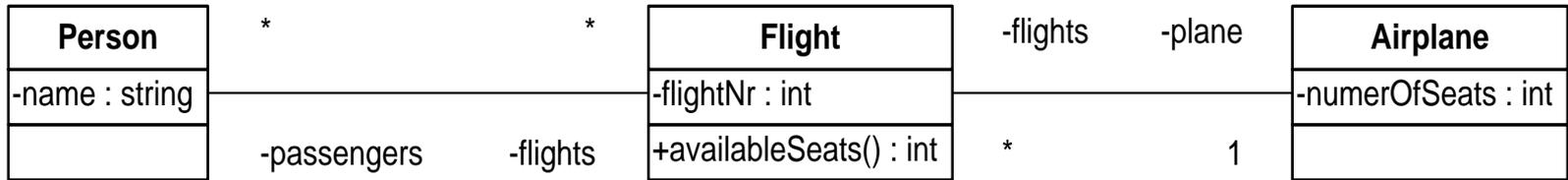
- **Invariant** : une contrainte sur un objet ou un groupe d'objets qui doit être **toujours vérifiée**
- Exemple :

```
context Compte  
inv : solde >= 0
```

Contraintes : invariants

- Une expression OCL est toujours évaluée pour une instance particulière
 - Nommage omit :
context Person
inv : age >= 18
 - Nommage par défaut : mot-clé *self*
context Person
inv : self.age >= 18
 - Nommage explicit :
context p : Person
inv : p.age >= 18

Accéder aux propriétés d'une classe



- Notation “.”
- Exemple: si Flight est le contexte, pour accéder :
 - à un attribut : `self.flightNr`
 - à une opération : `self.availableSeats()`
 - à l'autre côté de l'association : `self.plane`
- Noter l'importance de **rôles!**

Contraintes : pré/post conditions

- On peut spécifier les pré/post conditions pour les **opérations**
 - **pré-conditions** doit être vraie **avant** l'appel de l'opération
 - **post-conditions** doit être vraie **après** l'appel de l'opération
- Dans les post-conditions, on peut utiliser :
 - **result** : indiquer la valeur retournée par l'opération
 - **@pre** : indiquer la valeur d'un attribut avant l'appel de l'opération

Contraintes : exemple

```
context Compte::debiter(montant: Integer)
```

```
pre: montant > 0
```

```
post: solde = solde@pre - montant
```

```
context Compte::getSolde(): Integer
```

```
post: result = solde
```

Nommer les contraintes

- Syntaxe :

```
context class
  inv ConstraintName : constraintExpression
```

- Exemples :

```
context Compte
  inv soldePositif : self.solde > 0
```

```
context Compte::debiter(montant: Integer)
  pre montantPositif : montant > 0
  post montantDebite : self.solde = self.solde@pre -
montant
```

Commentaires

- Syntaxe :
 -- comment
- Exemples :

```
context Compte
inv : self.solde > 0  -- solde positif
```

```
context Compte::debiter(montant: Integer)
pre : montant > 0  -- montant positif
post montantDebite : self.solde = self.solde@pre
- montant
```

Résultat d'une opération

- Une expression OCL peut être utilisée pour indiquer le résultat d'une opération de type "query" :

```
context TypeName::operation(param1:Type1,...): retType
body:--expression qui retourne un objet de type retType
```

- Exemple :

```
context Compte::getSolde() : Float
body : solde
```

Valeurs initiales

- Une expression OCL peut être utilisée pour indiquer la valeur initiale d'un attribut

```
context TypeName::AttributeName: Type  
init: -- expression représentant la valeur initiale
```

- Exemple :

```
context Personne::marié : Boolean  
init: false
```

Valeurs dérivées

- Une expression OCL peut être utilisée pour indiquer la valeur dérivée d'un attribut

context TypeName::AttributeName: Type

derive:--expression représentant la règle de dérivation

- Exemple :

context Personne::age : Integer

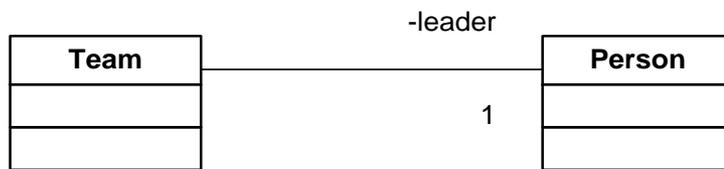
derive: Date::current() - dateDeNaissance

Rappel

- context Classe :: attribut : Type
init : valeur
- context Classe :: attribut : Type
derive : expression
- context Classe
inv : expressionBooléenne
- context Classe :: opération(paramètres) : Type
pre : expressionBooléenne
post : expressionBooléenne

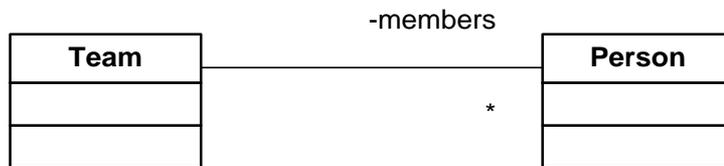
Navigabilité et Collections

- Dans la plupart du temps, le résultat d'une navigation n'est pas un seul objet mais une collection d'objets

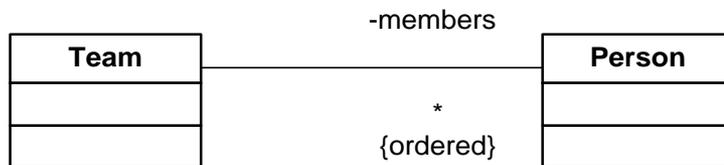


context Team

```
self.leader : Person
```

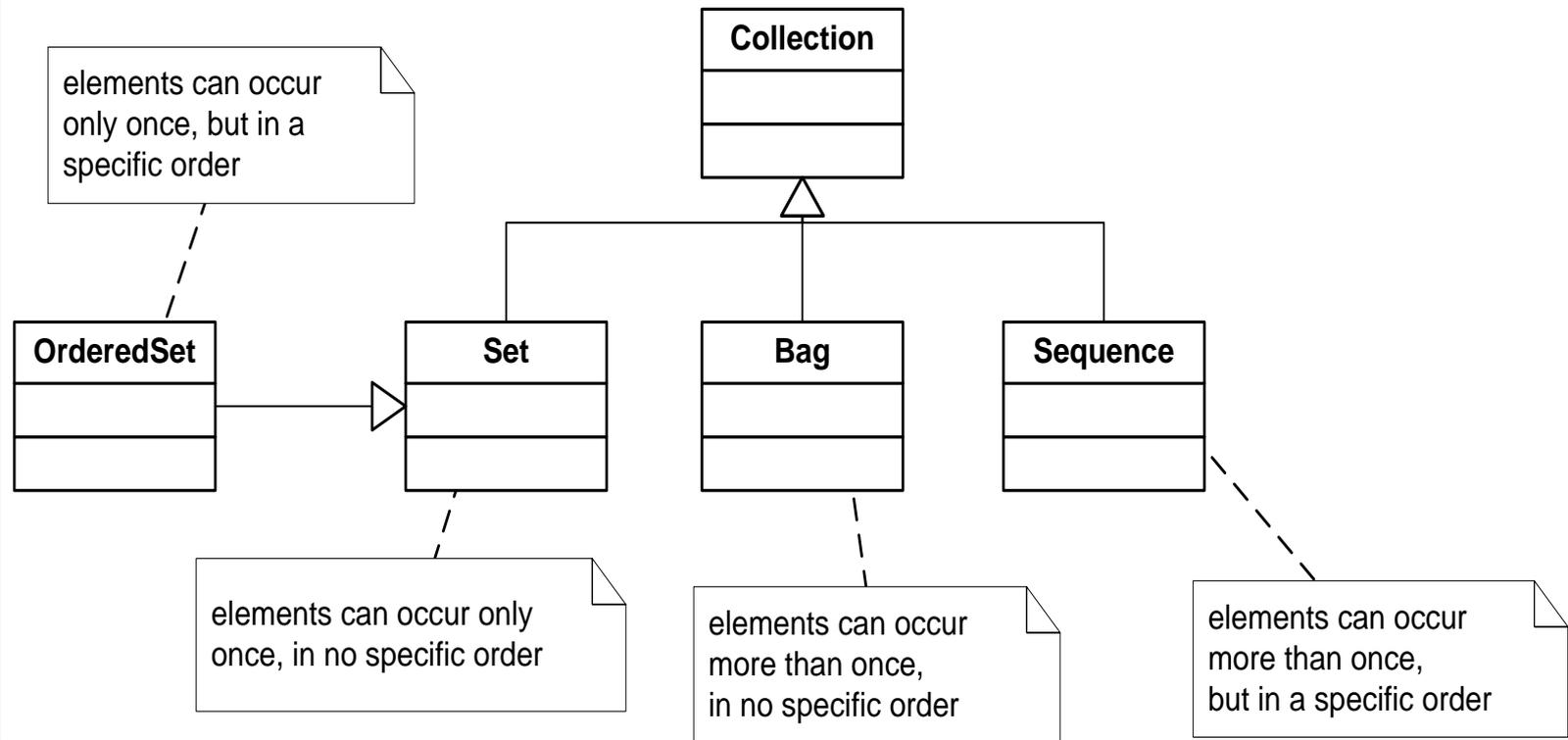


```
self.members : Set(Person)
```



```
self.members : OrderedSet(Person)
```

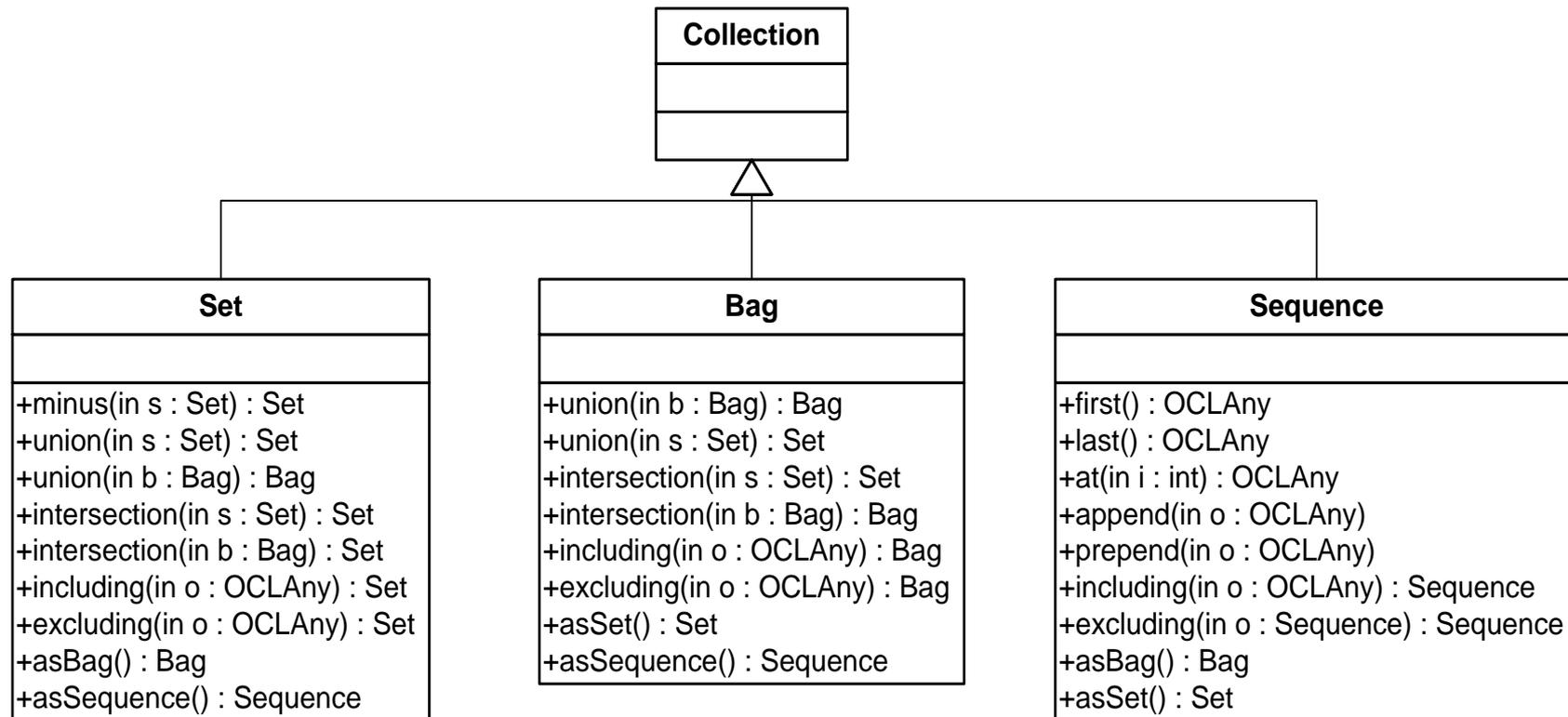
Hiérarchie de collections OCL



Opérations dans les toutes les collections

| Operation | Description |
|---|---|
| size() | The number of elements in the collection |
| count(object) | The number of occurrences of object in the collection. |
| includes(object) | True if the object is an element of the collection. |
| includesAll(collection) | True if all elements of the parameter collection are present in the current collection. |
| isEmpty() | True if the collection contains no elements. |
| notEmpty() | True if the collection contains one or more elements. |
| iterate(expression) | Expression is evaluated for every element in the collection. |
| sum(██████████) | The addition of all elements in the collection. |
| exists(expression) | True if expression is true for at least one element in the collection. |
| forAll(expression) | True if expression is true for all elements. |
| select(expression) | Returns the subset of elements that satisfy the expression |
| reject(expression) | Returns the subset of elements that do not satisfy the expression |
| collect(expression) | Collects all of the elements given by expression into a new collection |
| one(expression) | Returns true if exactly one element satisfies the expression |

Opérations spécialisées



Examples:

`Set{4,2,3,1}.minus(Set{2,3}) = Set{4,1}`

`Bag{1, 2, 3, 5}.including(6) = Bag{1, 2, 3, 5, 6}`

`Sequence{1, 2, 3, 4}.append(5) = Sequence{1, 2, 3, 4, 5}`

Opérations : exemples (I)

- `size() : Integer`
`context Company`
`inv : self.team->size() > 5`
- `isEmpty() : Boolean`
`context Person`
`inv : self.employer->isEmpty()`
- `notEmpty() : Boolean`
`context Company`
`inv : self.employees->notEmpty()`

Opérations : exemples (2)

- includes(object : T) : Boolean

```
context Bank
```

```
-- the owner of the bank is one of its clients
```

```
inv : self.clients->includes(self.owner)
```

- excludes(object : T) : Boolean

```
context Bank
```

```
-- the owner of the bank cannot be a client
```

```
inv : self.clients->excludes(self.owner)
```

Opérations : exemples (3)

- `count(object : T) : Boolean`

```
context ChessBoard
```

```
inv : self.pieces->count(k : Knight) = 4
```

- `sum() : T`

```
context Student
```

```
inv : self.notes->sum()/self.exams->size()>10
```

Opérations : exemples (4)

- `compte -> select(c | c.solde > 1000)`
- `compte -> reject(solde > 1000)`
- `compte -> collect(c : Compte | c.solde)`
- `(compte -> select(solde > 1000))-> collect(c|c.solde)`
- **context** Banque
inv: `not(clients -> exists(age < 18))`
- **context** Personne
inv: `Personne.allInstances() -> forAll(p1, p2 |
p1 <> p2 implies p1.nom <> p2.nom)`

Contraintes conditionnelles

- Syntaxe :
 - `if` `expr1` `then` `expr2` `else` `expr3` `endif`
 - `expr1` `implies` `expr2`

- Exemples :

```
context Personne
```

```
inv :    if age < 18 then compte->isEmpty()  
        else compte->notEmpty() endif
```

```
context Personne
```

```
inv: compte->notEmpty() implies banque->notEmpty()
```

Variables

- Les variables peuvent être utilisées pour améliorer la compréhension des contraintes complexes

- Syntaxe : **let** ... **in** ...

```
context Personne
```

```
inv: let argent = compte.solde->sum() in  
age >= 18 implies argent > 0
```

- Pour les rendre accessible partout : **def**

```
context Personne
```

```
def: argent : Integer = compte.solde->sum()
```

```
context Personne
```

```
inv: age >= 18 implies argent > 0
```