Ingénierie dirigée par les modèles (IDM)

Cours de 2e année ingénieur
TSI

Fabien.Romeo@eisti.fr

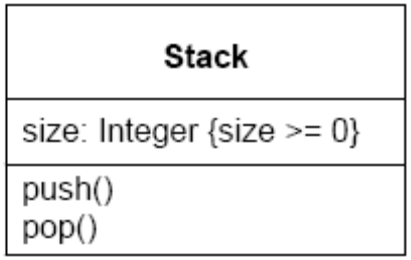
Contraintes UML

- Définition (Contrainte) :
  - Une contrainte est une restriction sur une ou plusieurs valeurs d'un modèle orienté objet
- Une contrainte peut être exprimée avec un langage formel (OCL) ou semi-formel ou en langage naturel
- Les contraintes sont écrites entre accolades :

  { contrainte }
- Les contraintes peuvent être écrites après un élément textuel de modélisation ou dans une note UML
- L'utilisateur peut définir ses propres contraintes :

<constraint> ::= '{' [<name> ':'] <Boolean-expression> ' }'

Exemples de contraintes



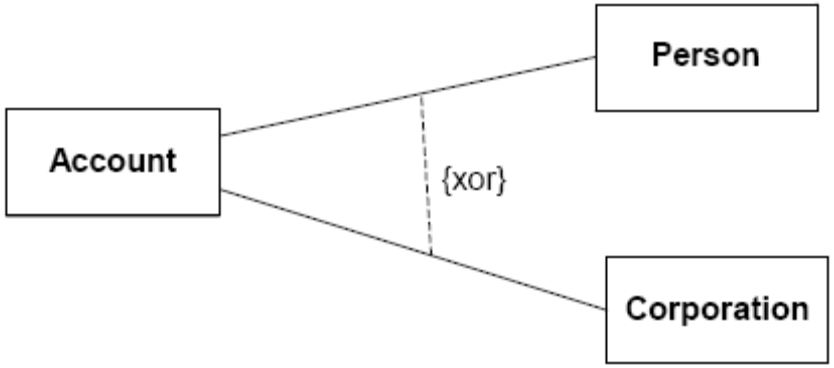


Figure 7.33 - Constraint attached to an attribute **Figure 7.34 - {xor} constraint**

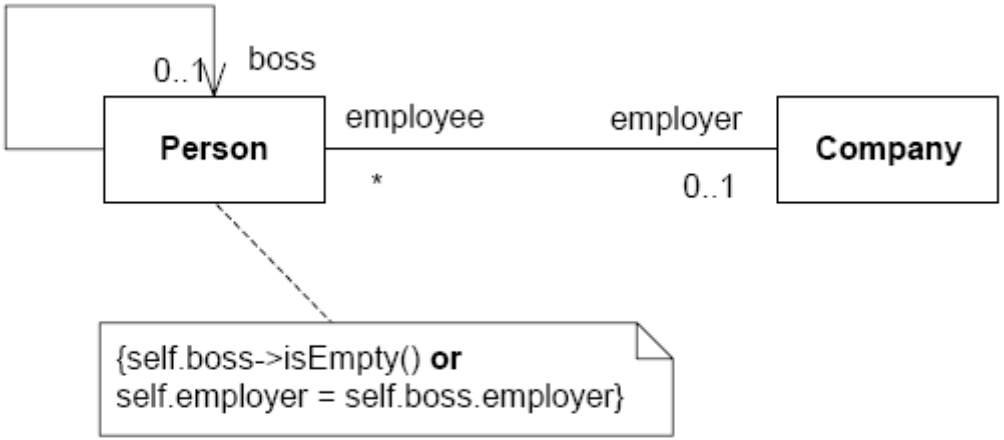
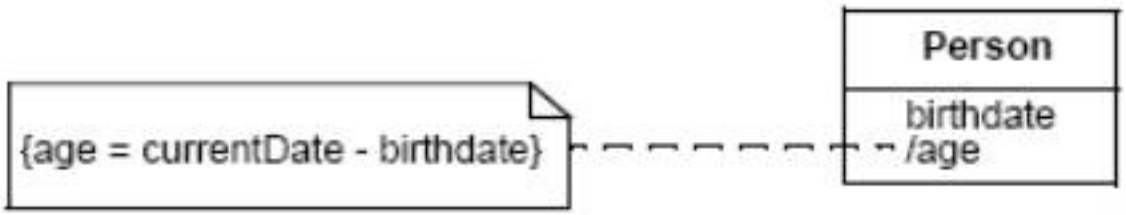
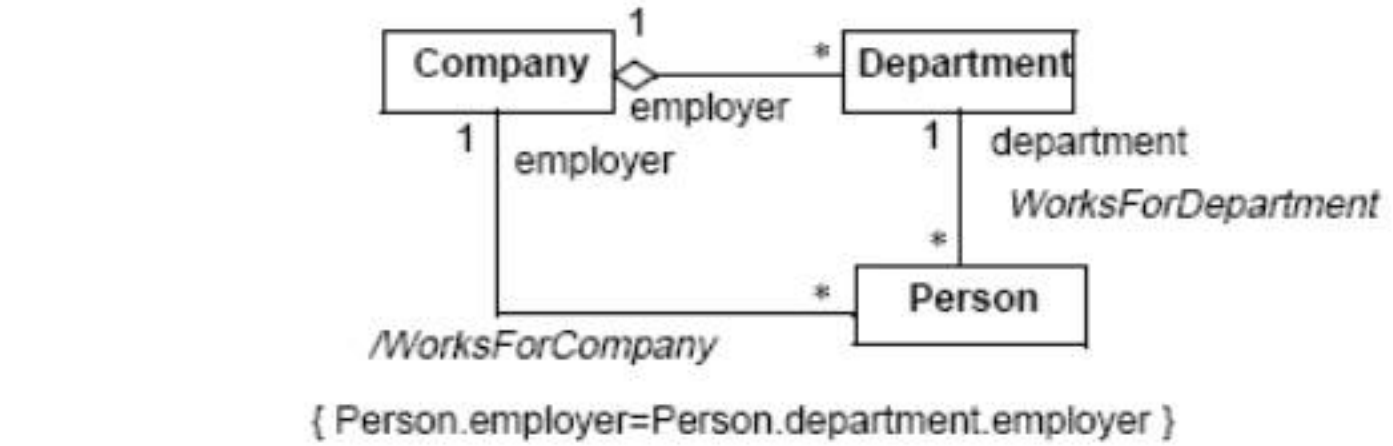


Figure 7.35 - Constraint in a note symbol

Contraintes et « derived »

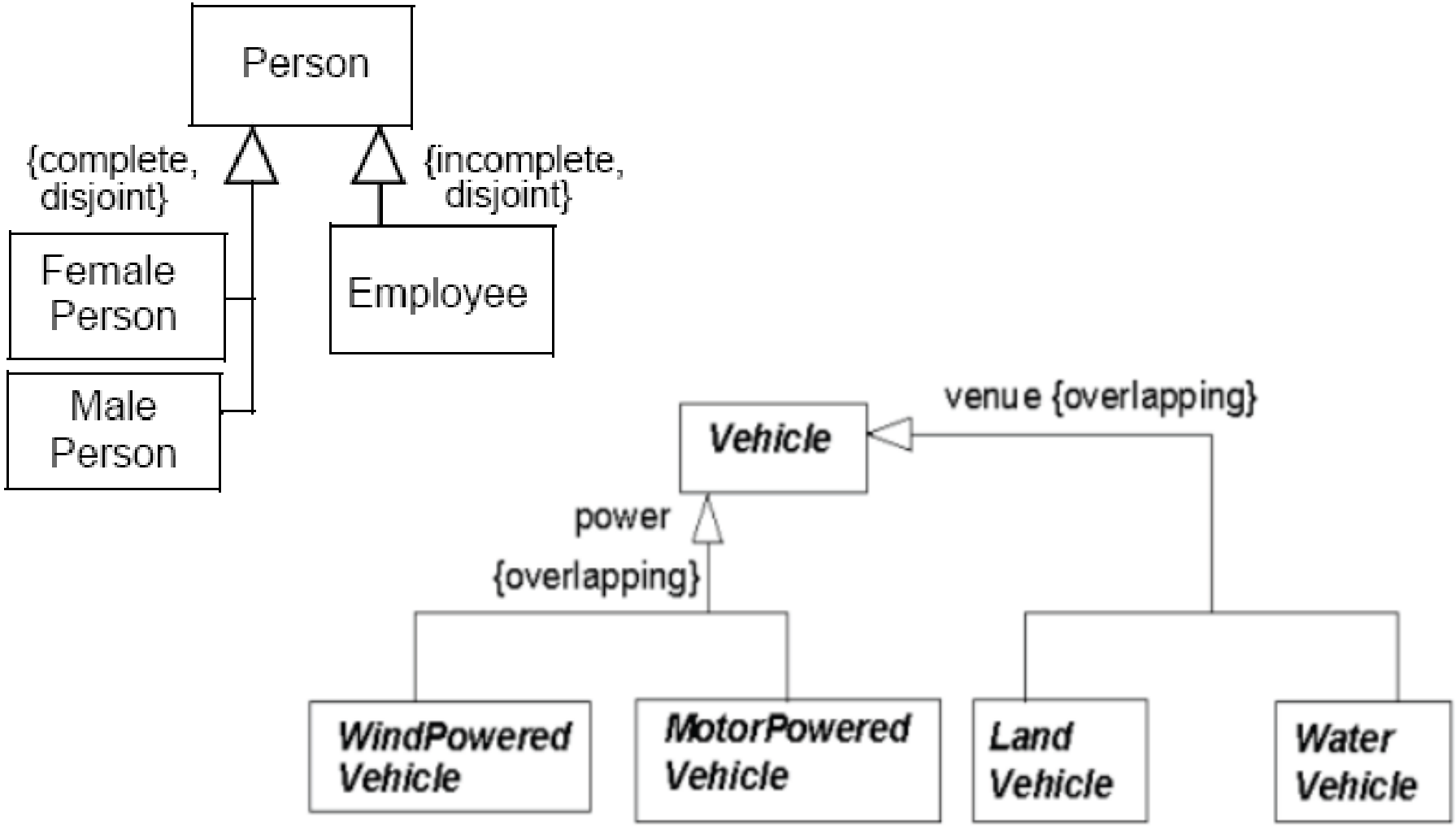




Contraintes de généralisation

- {complete, disjoint}
  - Généralisation terminée et pas d'instances communes.
- {incomplete, disjoint}
  - Généralisation extensible et pas d'instances communes.
- {complete, overlapping}
  - Généralisation terminée et instances communes possibles.
- {incomplete, overlapping}
  - Généralisation extensible et instances communes possibles.
- * par défaut {incomplete, disjoint}

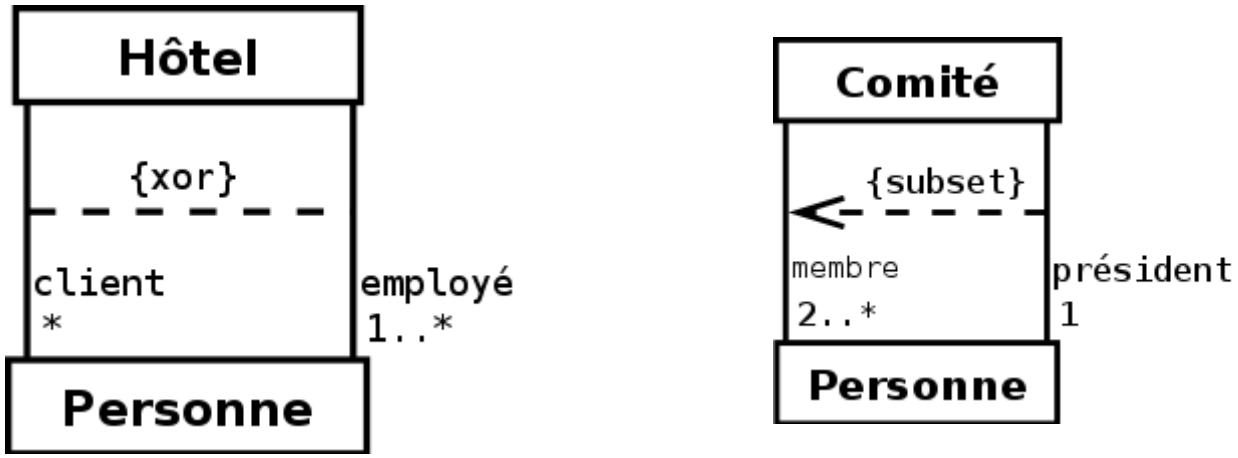
Contraintes de généralisation – Exemples



Contraintes d'association

- {subsets <property-name>}
  - Sous-ensemble de la propriété <property-name>.
- {redefines <end-name>}
  - Redéfinit <end-name>.
- {union}
  - Union des sous-ensembles.
- {ordered}
  - Ensemble ordonné (liste).
- {bag}
  - Collection sans ordre et avec doublons.
- {sequence} or {seq}
  - Sequence (bag ordonné).
- Si l'associationEnd est navigable, mêmes propriétés que sur un attribut.

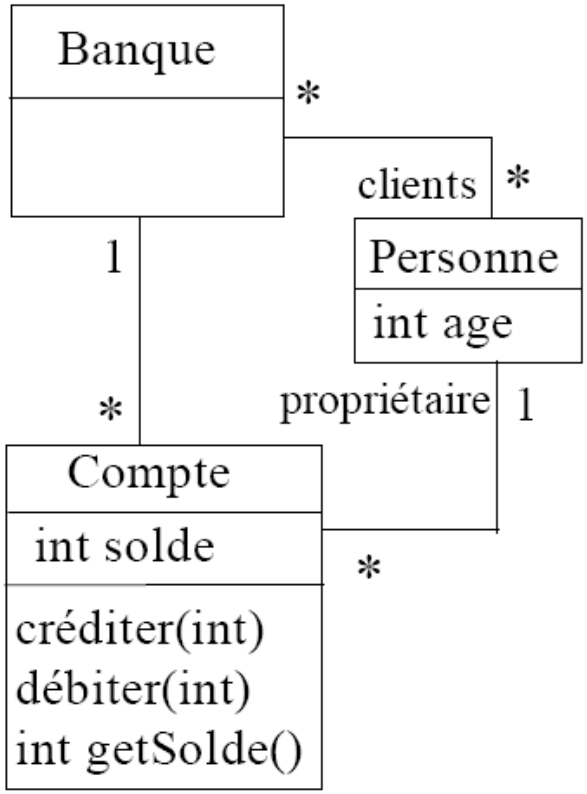
Contraintes d'association – Exemples



Contraintes OCL

- OCL = Object Constraint Language
- Fondations mathématiques mais pas de symboles mathématiques complexes
  - Utilisation de mots-clés : ex : forAll au lieu de ∀
- Langage fortement typé
  - Possibilité de vérification durant la modélisation sans « coder » le modèle
- Langage déclaratif
  - On reste au niveau de la modélisation sans se soucier des détails de l'implémentation

Context



- Le contexte d'une expression OCL correspond à l'élément de modèle (UML) à laquelle elle est attachée
  - Ex: classifier UML (classe, interface, …), opération, attribut, etc

- L'expression est évaluée pour une instance particulière : l'instance contextuelle
  - Nommage par défaut : mot-clé *self*

```
context Personne inv : self.age > 18
```

  - Nommage explicite :

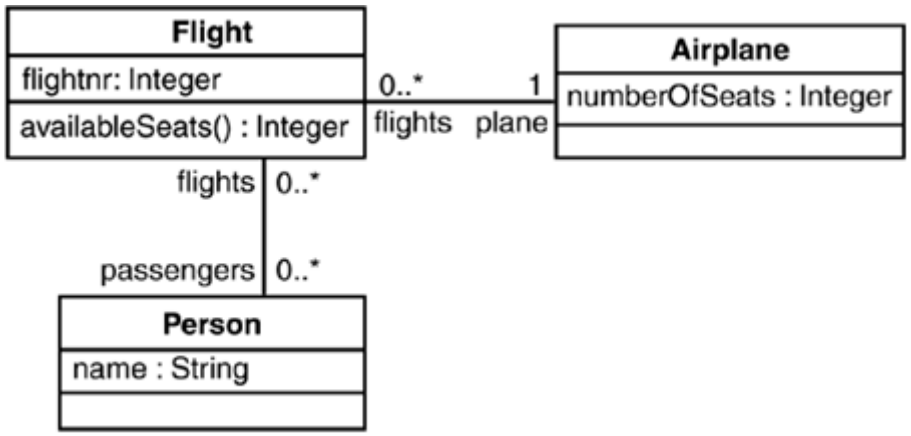
```
context p : Personne inv : p.age > 18
```

  - Omission :

```
context Personne inv : age > 18
```

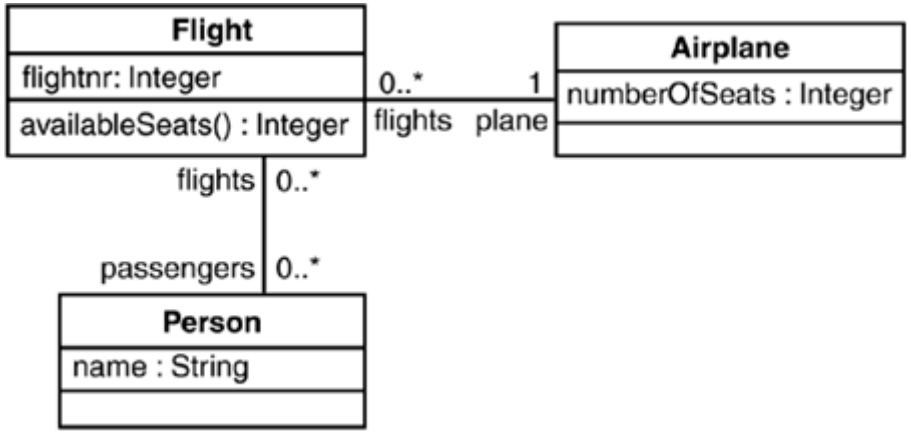
Accès aux propriétés d'un objet (navigation)

- Utilisation de la notation pointée « . »
- Ex : dans le contexte de Flight
  - Accès à un attribut : `self.flightnr`
  - Accès à une opération : `self.availableSeats()`
  - Accès une AssociationEnd opposée : `self.plane`
    - Importance du nommage des rôles pour la navigation !!!
    - S'il n'y a pas de nom au rôle : le nom du type avec une minuscule en 1e lettre est utilisé : `self.airplane` (si plane n'était pas défini)



Spécification d'une propriété

- Utilisation de la notation *deux points,deux points* « :: »
- Ex : dans le contexte de Flight
  - Attribut :
    `context Flight::flightnr : Integer`
  - Opération :
    `context Flight::availableSeats() : Integer`
  - AssociationEnd opposée :
    `context Flight::plane : Airplane`



Spécification de contrats

- Invariants
  - Un invariant exprime une contrainte sur un objet ou un groupe d'objets qui doit être respectée en permanence
  - Ex :

```
context Compte
inv: self.solde >= self.min
     and self.min <= 0
```

- Possibilité de combiner les invariants
  - Ex :

```
context Compte
inv: self.solde >= self.min
inv: self.min <= 0
```

Spécification de contrats

- Pre/Post conditions sur opérations
  - Précondition : état qui doit être respecté avant l'appel de l'opération. Mot-clé: `pre:`
  - Postcondition : état qui doit être respecté après l'appel. Mot-clé: `post:`
- Dans la postcondition, deux éléments particuliers sont utilisables
  - `result` : référence la valeur retournée par l'opération
  - `@pre` : référence une valeur avant l'appel

Spécification de contrats

- Exemples

```
context Compte::debiter(montant : Integer)
pre: montant > 0
     and montant <= self.solde - self.plancher
post: self.solde = self.solde@pre - montant

context Compte::getSolde() : Integer
post: result = self.solde

context Compte::crediter(montant : Integer)
pre: montant > 0
post: self.solde = self.getSolde()@pre + montant
```

Nommage des contraintes

- Syntaxe :
  - **context** Class
    inv ConstraintName : constraintExpression
- Exemples :
  - **context** Compte
    inv soldePositif : solde > 0
  - **context** Compte::debiter(montant : Interger)
    pre montantPositif : montant > 0
    post montantDebite : solde = solde@pre – montant

Commentaires

- Syntaxe :
  - **`-- commentaire`**
- Exemples :
  - ```
    context Compte
    inv: solde > 0 -- soldePositif
    ```
  - ```
    context Compte::debiter(montant : Interger)
    pre: montant > 0 -- montantPositif
    post: solde = solde@pre - montant -- montantDebite
    ```

Expressions de valeurs

- Spécification du corps d'une opération
  - Définit complètement le résultat d'une opération
    - Sorte de « post-condition exacte »
  - Seulement sur des « Query » opérations
    - Opérations sans effet de bord
    - Attribut *isQuery* du métamodèle UML

```
context Compte::getSolde() : Integer
body : self.solde
```

- Valeur initiale d'un attribut ou d'une AssociationEnd

```
context Compte::solde : Integer
init : 0
```

- Valeur dérivée d'un attribut ou d'une AssociationEnd

```
context Compte::interet : Integer
derive : self.solde * 0.05
```

Navigation et collections

- La plupart des navigations ne résulte pas en un objet unique mais en une collection


```
self.b : B
```


```
self.b : Set(B)
```


```
self.b : OrderedSet(B)
```

Collections OCL

- Collection(T) : super-type abstrait de tous les types collection typées OCL, template paramétré par le type T.
- Set(T) : type ensemble mathématique (pas d'occurrence multiple d'un élément)
- Bag(T) : multi-ensemble mathématique (possibilité d'occurrences multiples d'un élément)
- OrderedSet(T) : type ensemble ordonné (sans doublon, avec relation d'ordre)
- Sequence(T) : type suite (doublons possibles, relation d'ordre).
- Ex :

```
Set { 1 , 2 , 5 , 88 }
Set { 'apple' , 'orange', 'strawberry' }
OrderedSet { 'apple' , 'orange', 'strawberry', 'pear' }
Sequence { 1, 3, 45, 2, 3 }
Sequence { 'ape', 'nut' }
Bag {1 , 3 , 4, 3, 5 }
```

Quelques opérations sur Collection (notation ->)

- `size() : Integer`
- `includes(object : T) : Boolean`
- `excludes(object : T) : Boolean`
- `count(object : T) : Integer`
- `includesAll(c2 : Collection(T)) : Boolean`
- `excludesAll(c2 : Collection(T)) : Boolean`
- `isEmpty() : Boolean`
- `notEmpty() : Boolean`
- `sum() : T`

Quelques opérations sur Set

- `union(s : Set(T)) : Set(T)`
  et aussi intersection, -, etc
- `union(bag : Bag(T)) : Bag(T)`
  et aussi intersection
- `= (s : Set(T)) : Boolean`
- `including(object : T) : Set(T)`
- `excluding(object : T) : Set(T)`
- `flatten() : Set(T2)`
- `count(object : T) : Integer`
- conversion vers Bag, …

Quelques opérations sur OrderedSet

- `append (object: T) : OrderedSet(T)`
- `prepend(object : T) : OrderedSet(T)`
- `insertAt(index : Integer,`
  `object : T) : OrderedSet(T)`
- `at(i : Integer) : T`
- `indexOf(obj : T) : Integer`
- `first() : T`
- `last() : T`

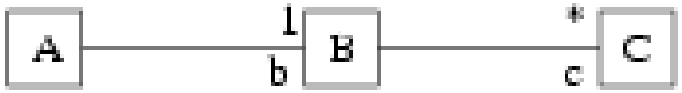
Navigation et collections

- Cas particulier
  - Cardinalité 0..1 traitée comme une collection pour tester l'existance d'une référence
    - Transformer la référence en Set (singleton ou ensemble vide) : utiliser ->asSet() ;
    - Test à vide de cet ensemble
    - ->asSet() implicite (facilité syntaxique)

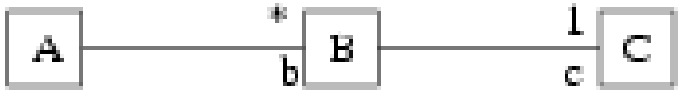
```
context Account
inv : bankBook->notEmpty() implies owner.age <= 28
-- inv : bankBook->asSet()->notEmpty() implies
   owner.age <= 28
```

Navigations enchaînées

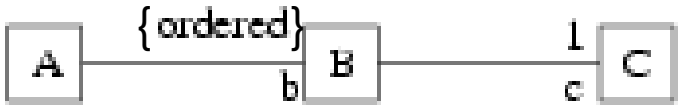
- Attention au typage !



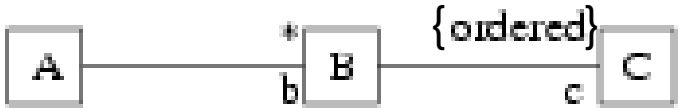

```
self.b.c : Set(C)
```



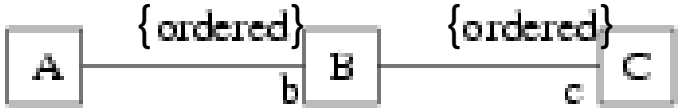

```
self.b.c : Bag(C)
```




```
self.b.c : Sequence(C)
```

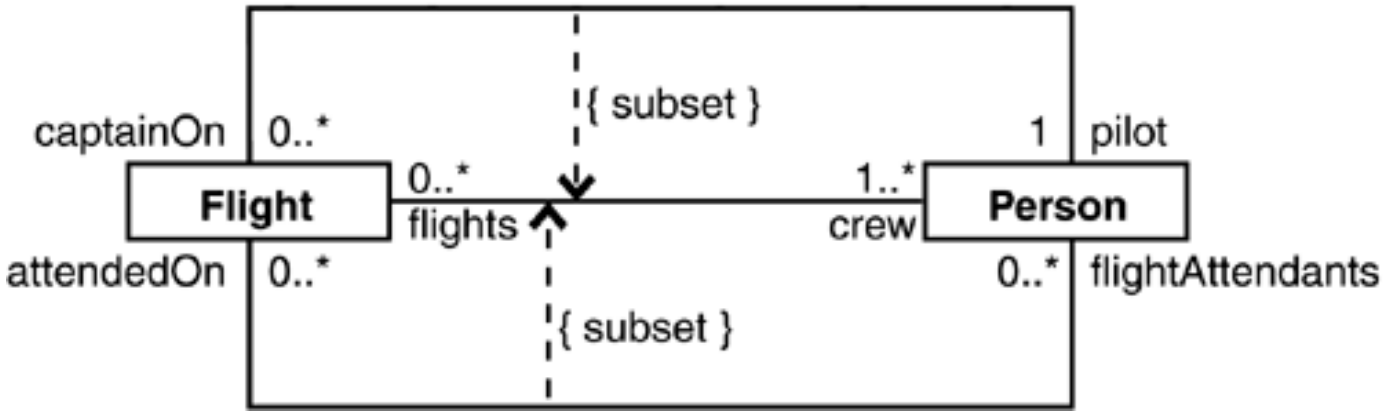



```
self.b.c : Bag(C)
```




```
self.b.c : Sequence(C)
```

Contraintes UML vs OCL




```
context Flight
 inv: self.crew->includes(self.pilot)
 inv: self.crew->includesAll(self.flightAttendants)

context Person
 inv: self.flights->includes(self.captainOn)
 inv: self.flights->includesAll(self.attendedOn)
```

ltérateurs sur collection

- any(expr) : un élt aléatoire parmi ceux qui vérifient expr
- collect(expr) : collection des objets qui vérifient expr
- collectNested(expr) : collection de collections
- exists(expr) : vrai si au moins un élt vérifie expr
- forAll(expr) : vrai si tous les élts vérifient expr
- isUnique(expr) : vrai si expr à une seule valeur
- iterate(...) : itère sur les élts de la collection
- one(expr) : vrai si seulement un élt vérifie expr
- reject(expr) : sous-collection des élts ne vérifiant pas expr
- select(expr) : sous-collection des élts vérifiant expr
- sortedBy(expr) : collection ordonnée des élts suivant expr

Itérateurs sur collection - Exemples

- Dans le contexte de la classe Banque
  - `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)`

Conditionnelles

- Certaines contraintes sont dépendantes d'autres contraintes. Deux formes pour gérer cela :
  - **if** expr1 **then** expr2 **else** expr3 **endif** : si l'expression expr1 est vraie alors expr2 doit être vraie sinon expr3 doit être vraie
  - expr1 **implies** expr2 : si l'expression expr1 est vraie, alors expr2 doit être vraie également. Si expr1 est fausse, alors l'expression complète est vraie

Conditionnelles - Exemples

- **context** Personne **inv:**

  if age < 18

  then compte -> isEmpty()

  else compte -> notEmpty()

  endif

- **context** Personne **inv:**

  compte -> notEmpty() **implies**
  banque -> notEmpty()

Variables

- Pour faciliter l'utilisation de certains attributs ou calculs de valeurs on peut définir des variables
- Dans une contrainte OCL : `let ... in ...`

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

- Pour l'utiliser partout : `def`

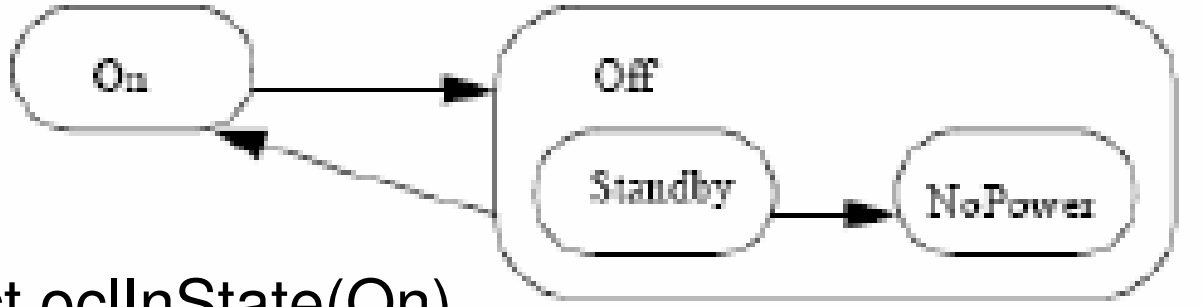
```
context Personne
def: argent : int = compte.solde -> sum()
context Personne
inv: age >= 18 implies argent > 0
```

Opérations sur les objets

- oclIsTypeOf(t : OclType) : Boolean
  - Le résultat est vrai si le type de *self* et *t* sont identiques :
    context Person
    inv : self.oclIsTypeOf(Person) -- is true
    inv : self.oclIsTypeOf(Company) -- is false
- oclIsKindOf(t : OclType) : Boolean
  - Vrai si t est le type direct ou un des supertypes de l'objet considéré
- oclIsNew() : Boolean
  - Uniquement dans les post-conditions
  - Vrai si l'objet a été créé au cours de l'exécution de l'opération (*i.e.* n'existe pas en précondition)

Opérations sur les objets (2)

- oclIsInState(s : OclState) : Boolean
  - Vrai si l'objet est dans l'état s
  - s correspond à un état d'une machine à états associée au classifier



  - object.oclInState(On)
  - object.oclInState(Off)
  - object.oclInState(Off::Standby)
  - object.oclInState(On::NoPower)

Messages

- Type : OclMessage
- HasSent operator : ‘ ^ ’
  - **context** Subject::hasChanged()
    post: observer^update(12,14)
  - **context** Subject::hasChanged()
    post: observer^update(? : Integer,? : Integer)
- Sequence de messages : ‘ ^^ ’
  - **context** Subject::hasChanged()
    post: observer^^update(12,14)
  - **context** Subject::hasChanged()
    post: let messages : Sequence(OclMessage) =
    observer^^update (? : Integer,? : Integer) **in**
    messages->notEmpty() **and**
    messages->exists(m | m.i > 0 **and** m.j >= m.i)

Outils

Dresden OCL2 for Eclipse

- Eclipse 3.4
- Plugin EMF (modeling and stuff)
- Plugin PDE
- Plugin Subclipse
- Via SVN : https://dresden-ocl.svn.sourceforge.net/svnroot/dresden-ocl/

EMF with MDT OCL

- Christian W. Damus, IBM Rational Software « Implementing Model Integrity in EMF with MDT OCL », Feb. 2007
  - http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html

Autres outils

- http://www.klasse.nl/ocl/ocl-tools.html