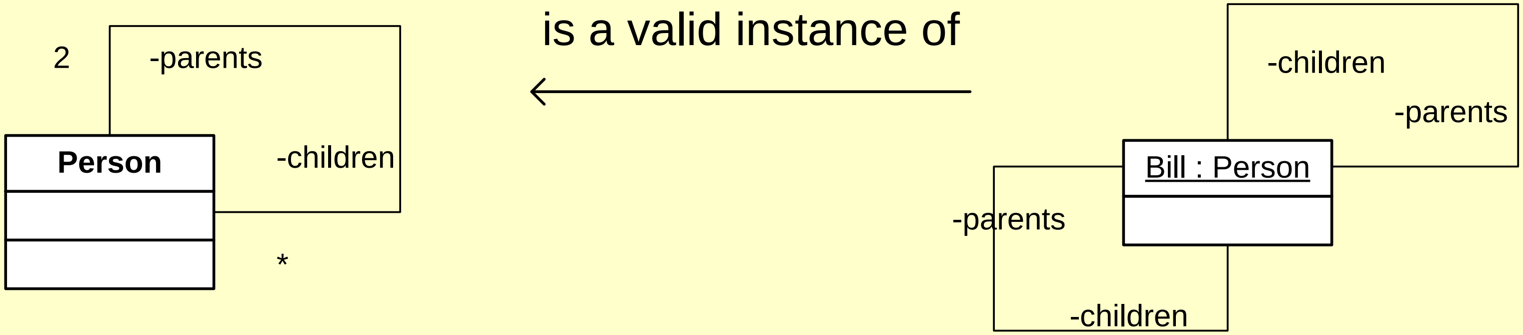
Object Constraint Language

OCL 2.0

UML does not tell us everything!



2

-parents

Person

-children

*

is a valid instance of

Bill : Person

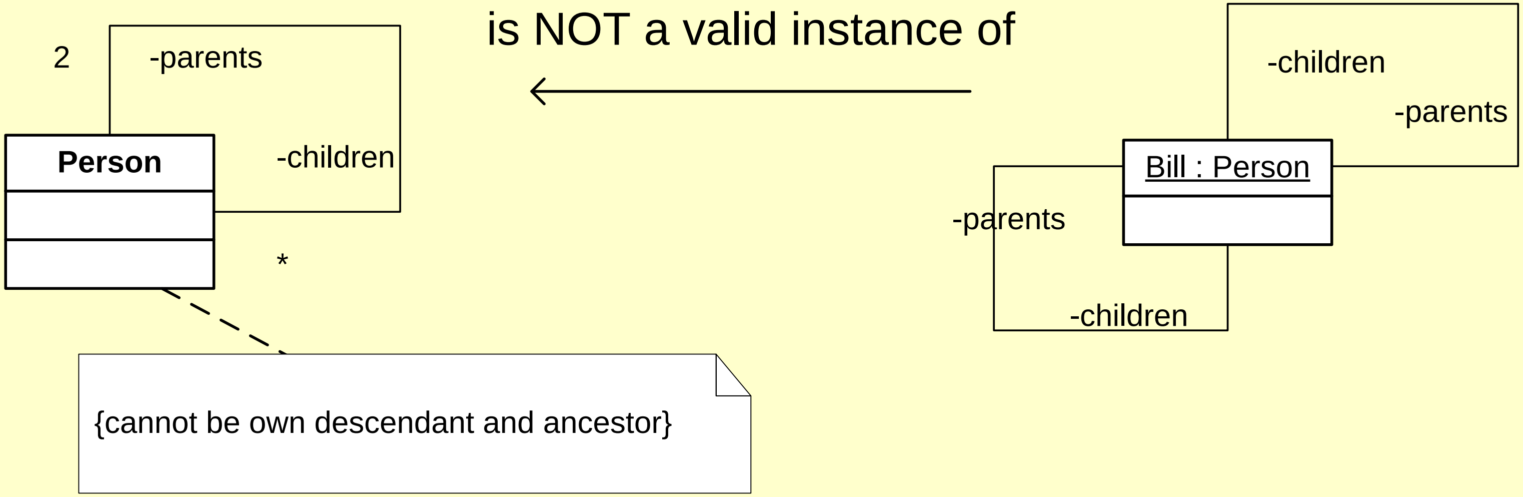
-children

-parents

-parents

-children

Enter constraints



is NOT a valid instance of

{cannot be own descendant and ancestor}

What is OCL?

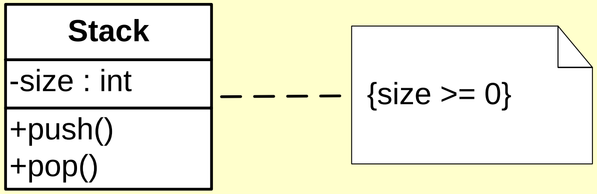
- A language to express constraints in our UML models
- A precise and unambiguous language that can be read and understood by developers and customers
- A purely declarative language: it describes *what* and not *how*

What is an OCL constraint?

- An OCL expression that evaluates to true or false
- Constraints are expressed as : {constraint}
- Put after text elements in a UML diagram, or in a note
- Constraints can be of three kinds:
  - Invariants
  - Pre-conditions
  - Post-conditions

  More on this later…

OCL constraints: examples

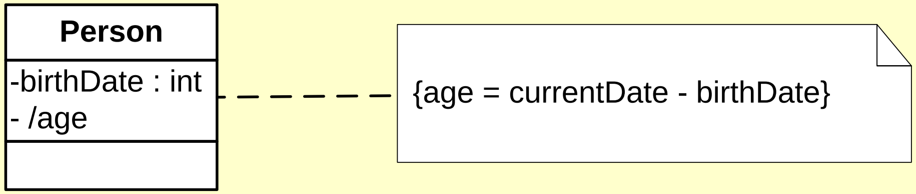


Stack

-size : int

+push()
+pop()

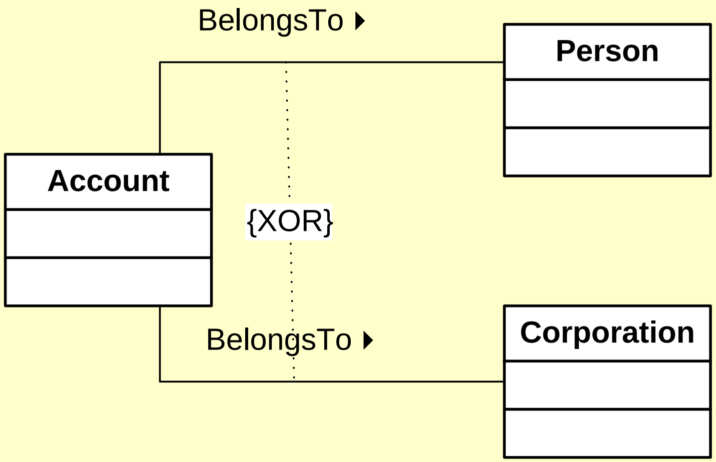
{size >= 0}



Person

-birthDate : int
- /age

{age = currentDate - birthDate}



BelongsTo ▸

Person

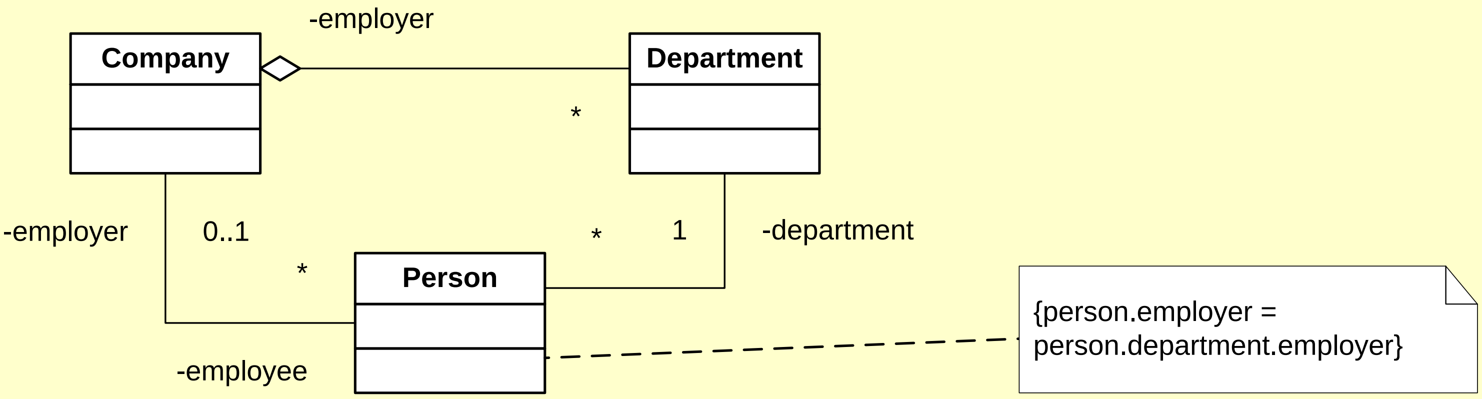
Account

{XOR}

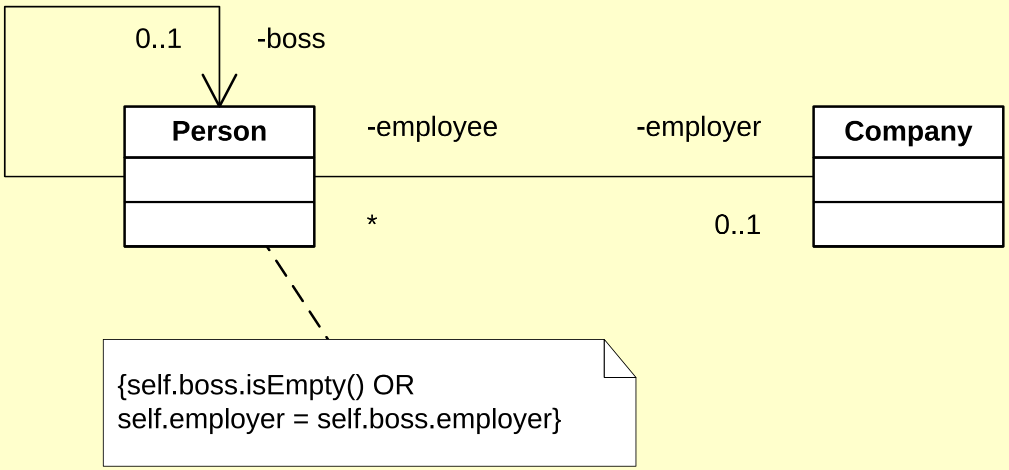
BelongsTo ▸

Corporation

OCL constraints: more examples



{person.employer =
person.department.employer}

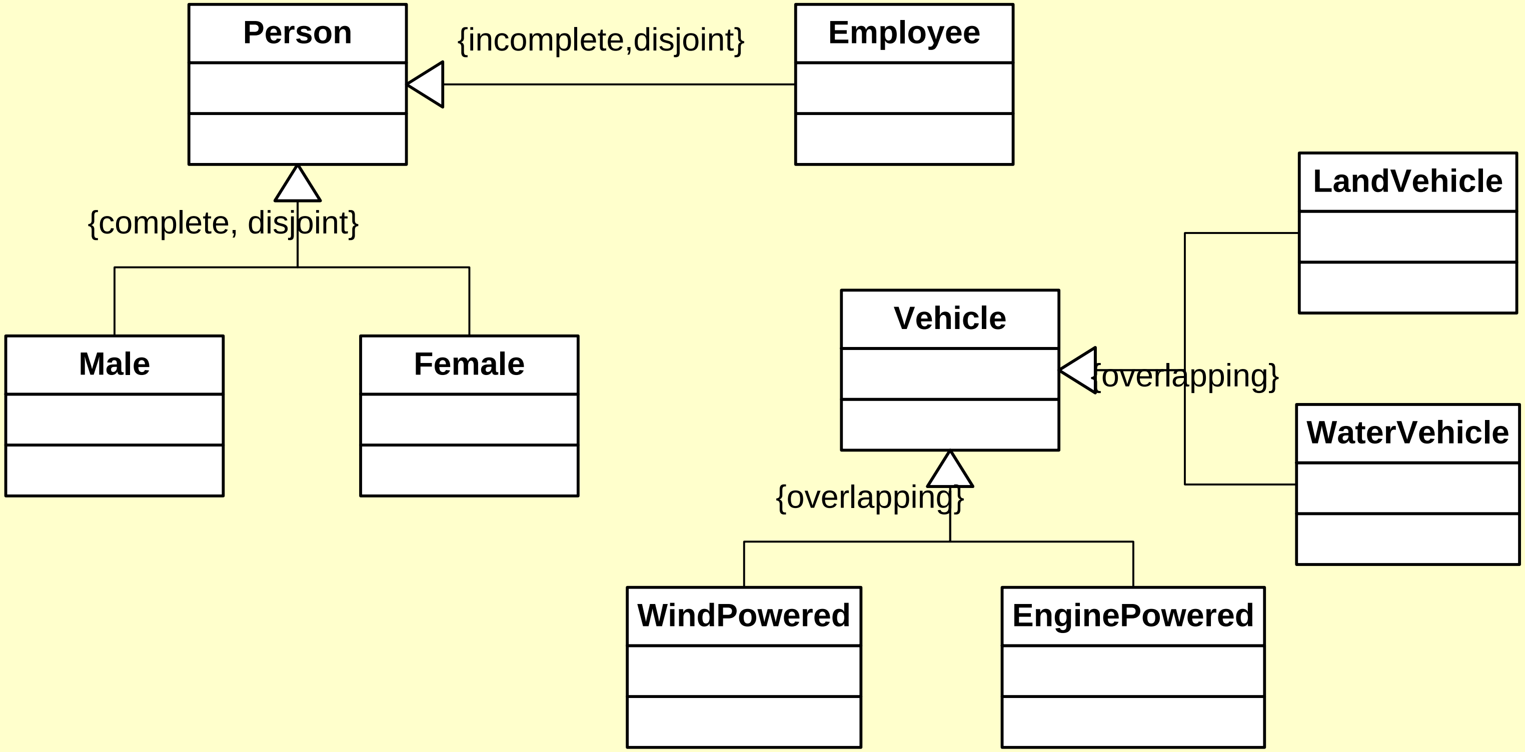


{self.boss.isEmpty() OR
self.employer = self.boss.employer}

Generalization constraints

- {complete, disjoint}
  - Not extensible, with no common instances
- {incomplete, disjoint}
  - Extensible, with no common instances
- {complete, overlapping}
  - Not extensible, with common instances
- {incomplete, overlapping}
  - Extensible, with common instances
- By default : {incomplete, disjoint}

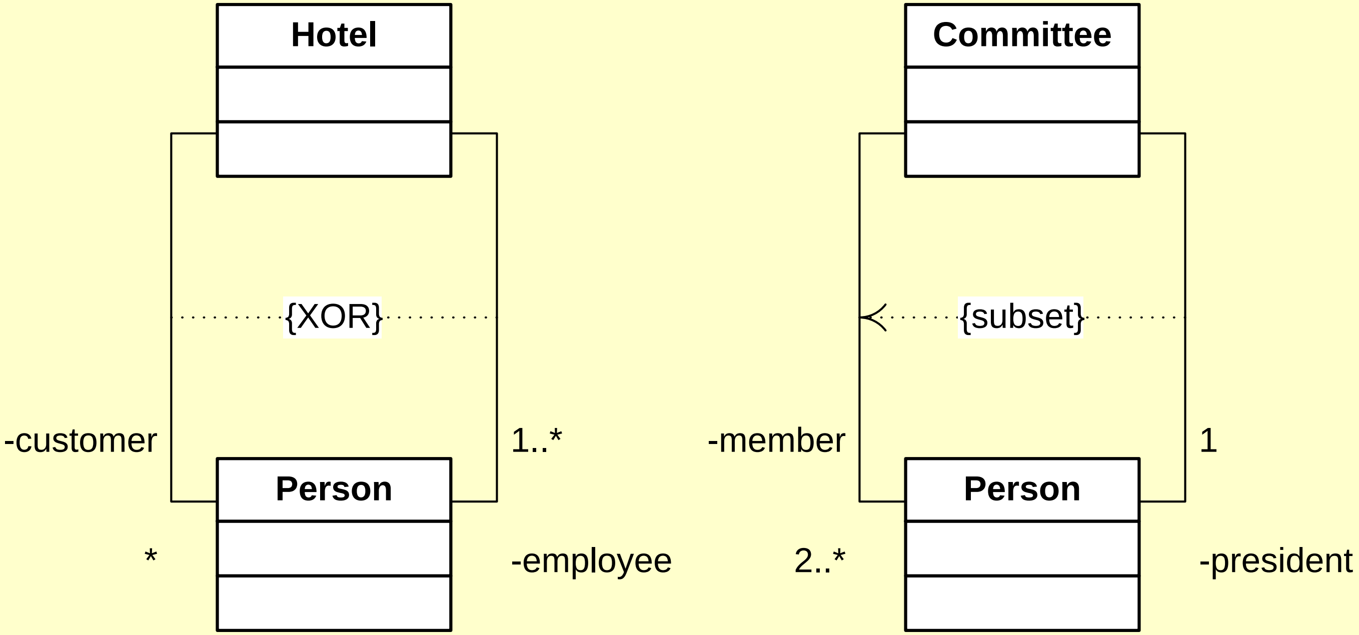
Generalization constraints: examples



Association constraints

- {subsets <property_name>}
- {redefines <property_name>}
- {union}
- {ordered}
- {bag}
- {sequence} or {seq}

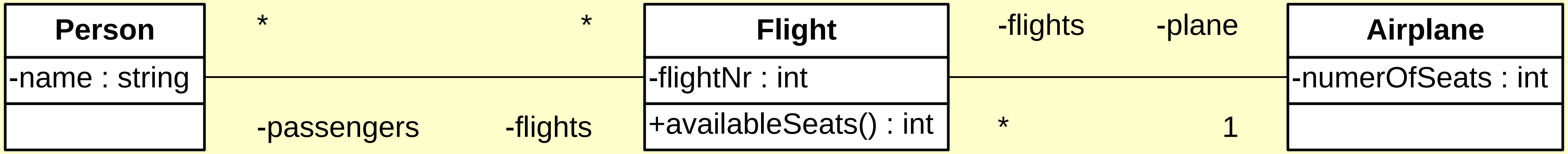
Association constraints examples



OCL: Context

- The context of an OCL expression is the UML element (class, attribute, relation, …) to which it is attached
- An OCL expression is always evaluated for a particular instance (the contextual instance)
  - Default naming: *self* keyword
    - `context Person inv : self.age >= 18`
  - Explicit naming:
    - `context p : Person inv : p.age >= 18`
  - Omitted
    - `context Person inv : age >= 18`

Accessing Class Properties



- Dot "." notation is used
- Example: if Flight is the context, to access:
  - an attribute: self.flightNr
  - an operation: self.availableSeats()
  - the opposite association end: self.plane
- Note the importance of roles!

Property Specification

- Double-colon notation “::”
- Example: if Flight is the context,
  - For an attribute:

```
context Flight::flightNb : int
```

  - For an operation:

```
context Flight::availableSeats() : int
```

  - For an association end

```
context Flight::plane : Airplane
```

Constraints: Invariants

- Invariant: a constraint on a (group of) object(s) which must be always verified

```
context Account
inv: self.balance >= self.min AND self.min >= 0
```

- Invariants can be combined:

```
context Account
inv: self.balance >= self.min
inv: self.min >= 0
```

Constraints: Pre/Post conditions

- In OCL we can specify pre/post conditions for operations
  - Pre-conditions: must be verified before operation call
  - Post-conditions: must be verified after operation call

- In post-conditions, two specific elements can be accessed
  - result: refers to the value returned by the operation
  - @pre: refers to the value of an attribute before the call

Constraints: an Example

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

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

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

Naming Constraints

- Syntax:

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

- Examples

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

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

Comments

- Syntax:

```
-- comment
```

- Examples

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

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

Operation Body Expression

- An OCL expression can be used to indicate the result of a query operation

```
context TypeName::operationName(param1 : Type1, …): retType
body: -- an expression returning an object of type retType
```

- Example

```
context Person::getCurrentSpouse(): Person
pre: self.isMarried = true
body: self.marriages->select( m | m.ended = false).spouse
```

Initial or Derived Values

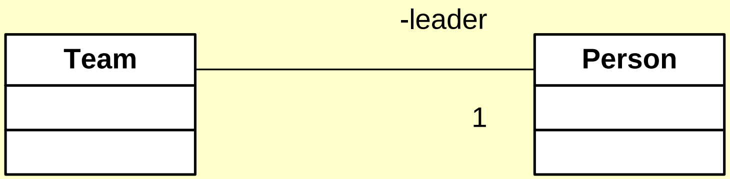
- An OCL expression can be used to indicate the initial or derived value of an attribute or association end
  - **context** TypeName::AttributeName: Type
    init: -- some expression representing the initial value
  - **context** TypeName::AttributeName: Type
    derive: -- some expression representing the derivation rule
- Examples

```
context Person::income : int
init: 0

context Person::age : int
derive: currentDate – self.birthdate
```

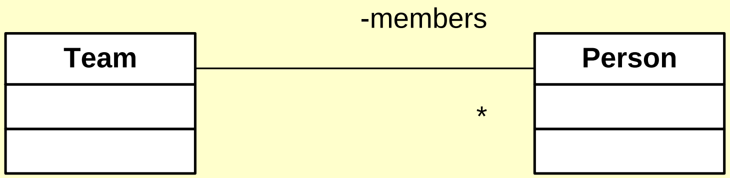
Navigability & Collections

Most of the time, the result of a navigation is not a single object, but a collection of objects

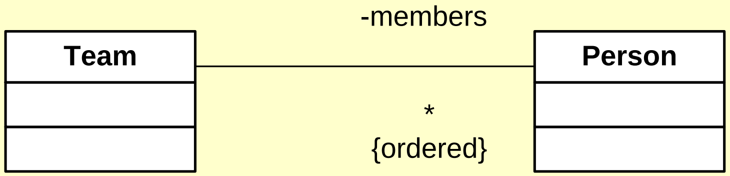
```
context Team
```



```
self.leader : Person
```

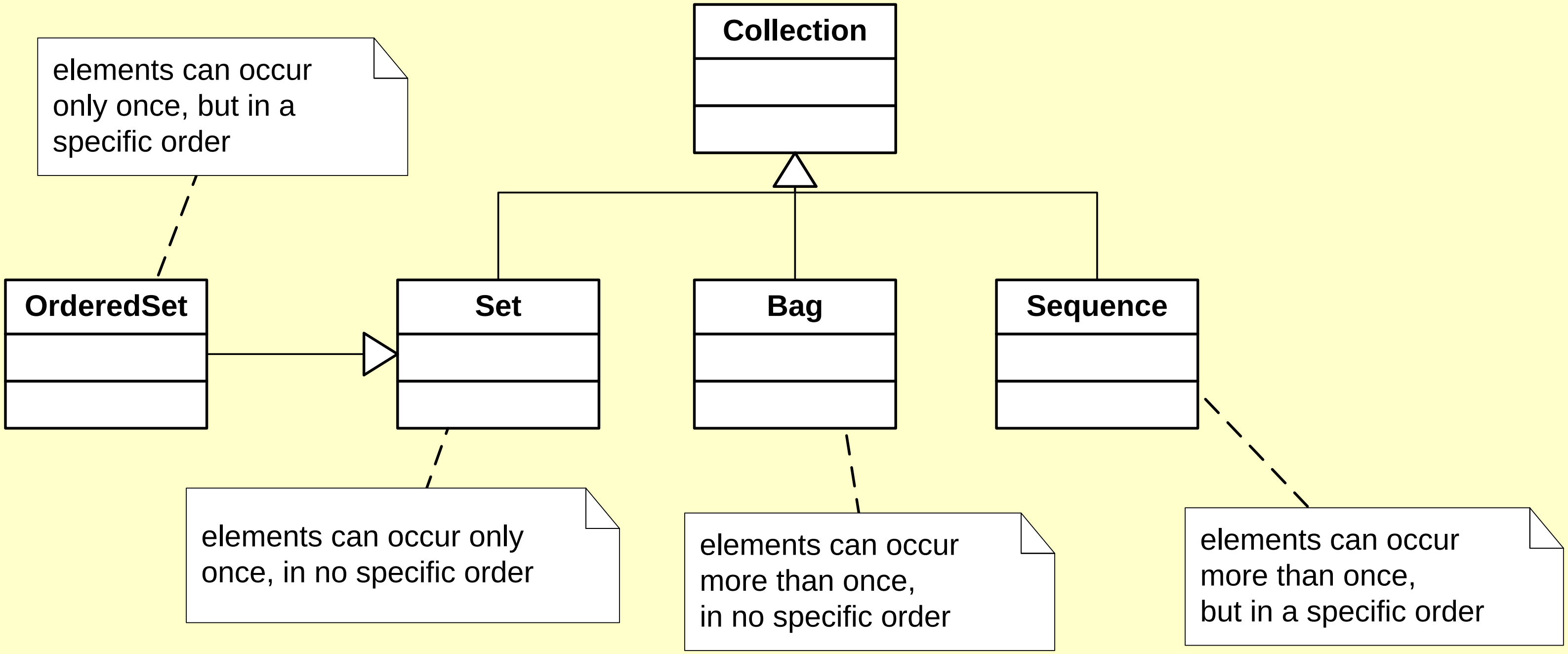


```
self.members : Set(Person)
```



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

The OCL Collection Hierarchy



elements can occur only once, but in a specific order

elements can occur only once, in no specific order

elements can occur more than once, in no specific order

elements can occur more than once, but in a specific order

Operations on All Collections

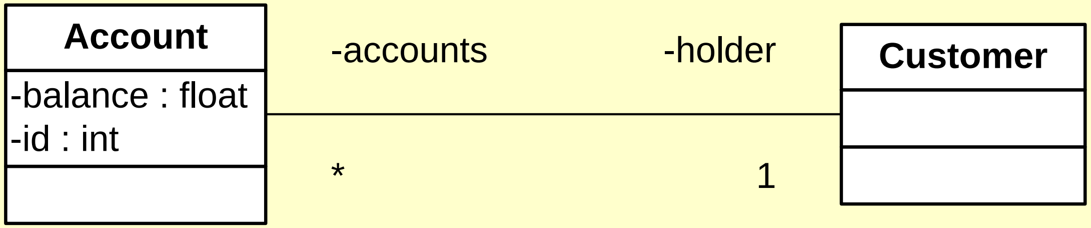
Operation	Description	
size()	The number of elements in the collection	
count(object)	The number of occurences of object in the collection.	
includes(object)	True if the object is an element of the collection.	
includesAll(collection)	True if all elements of the parameter collection are present in the 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(collection)	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	
sortedBy(expression)	Returns a Sequence of all the elements in the collection in the order specified (expression must containt the < operator	
any (expression)	A random element which satisfies expression	

The “->” Notation

- Operations on collections are introduced by “->”
- Example:

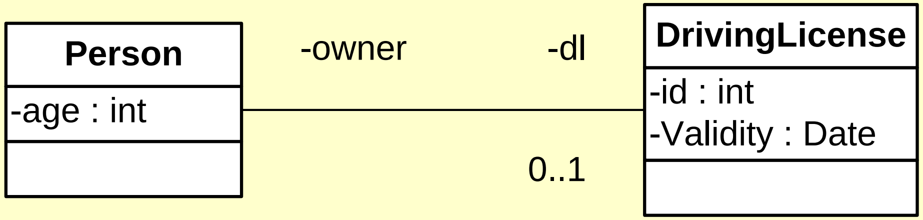
```
context Person p
inv : p.children->size() >= 0

context Company c
inv : not c.employees->isEmpty()
```

The “Select” Operator on Collections



- Customer.accounts.balance = 0 is not allowed
- Customer.accounts->select(id=2324).balance = 0 is allowed

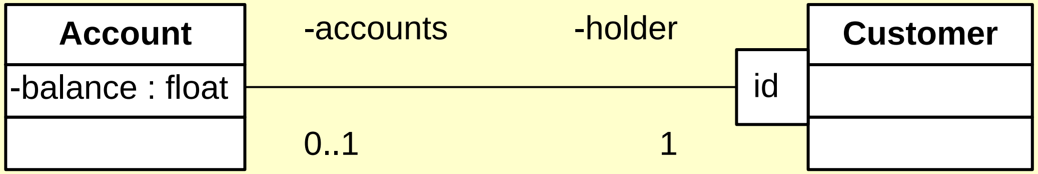
Special Case for Collections



- 0..1 multiplicity end is considered as a collection to test the existence of an element
- Implicit ->asSet() operator to simplify notation

```
context Person p
inv : p.dl->notEmpty() implies p.age >= 18
-- more precisely : p.dl->asSet()->notEmpty()…
```

Navigating across Qualified Associations

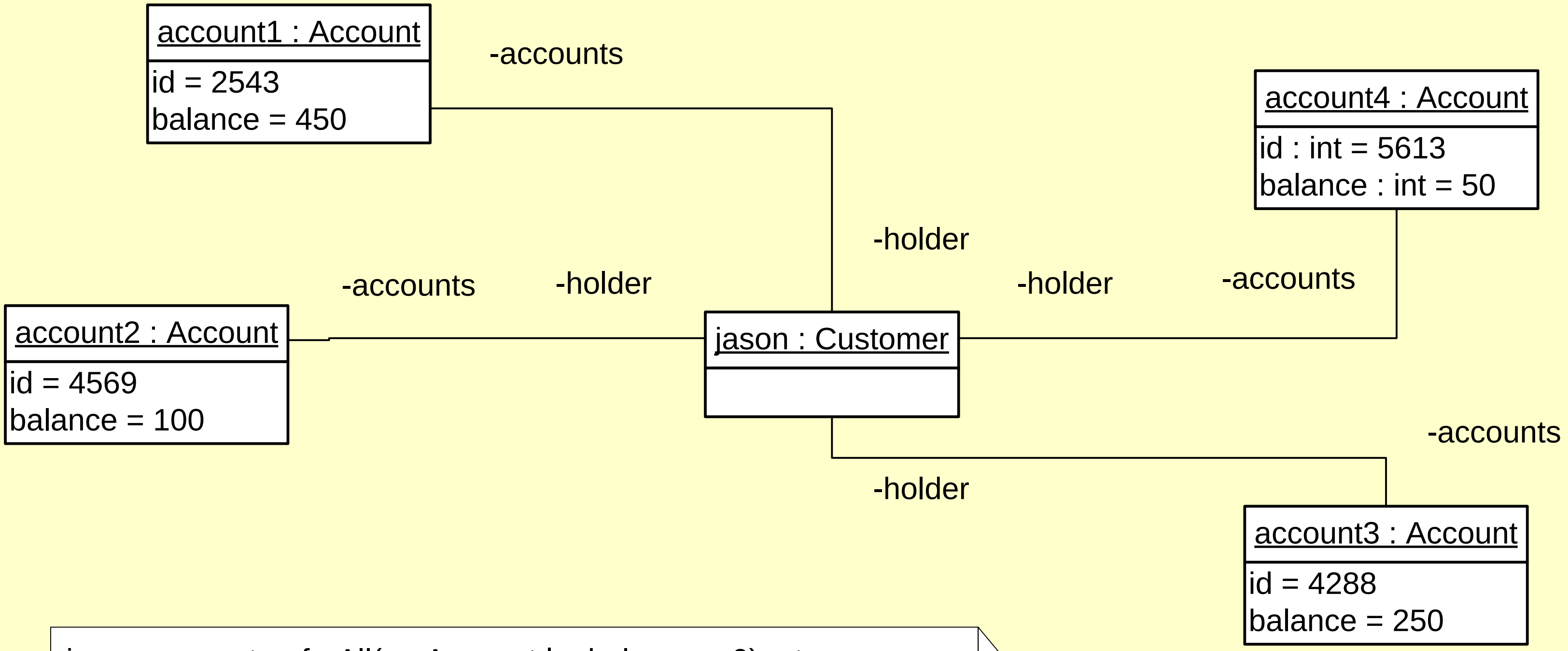


In OCL, to access a qualified end:

- customer.accounts[3254] or
- customer.accounts[id=3254]

NB: id is an attribute of Account class

Operations on All Collections



account1 : Account

id = 2543
balance = 450

-accounts

account4 : Account

id : int = 5613
balance : int = 50

-holder

-accounts

-holder

-holder

-accounts

account2 : Account

id = 4569
balance = 100

jason : Customer

-accounts

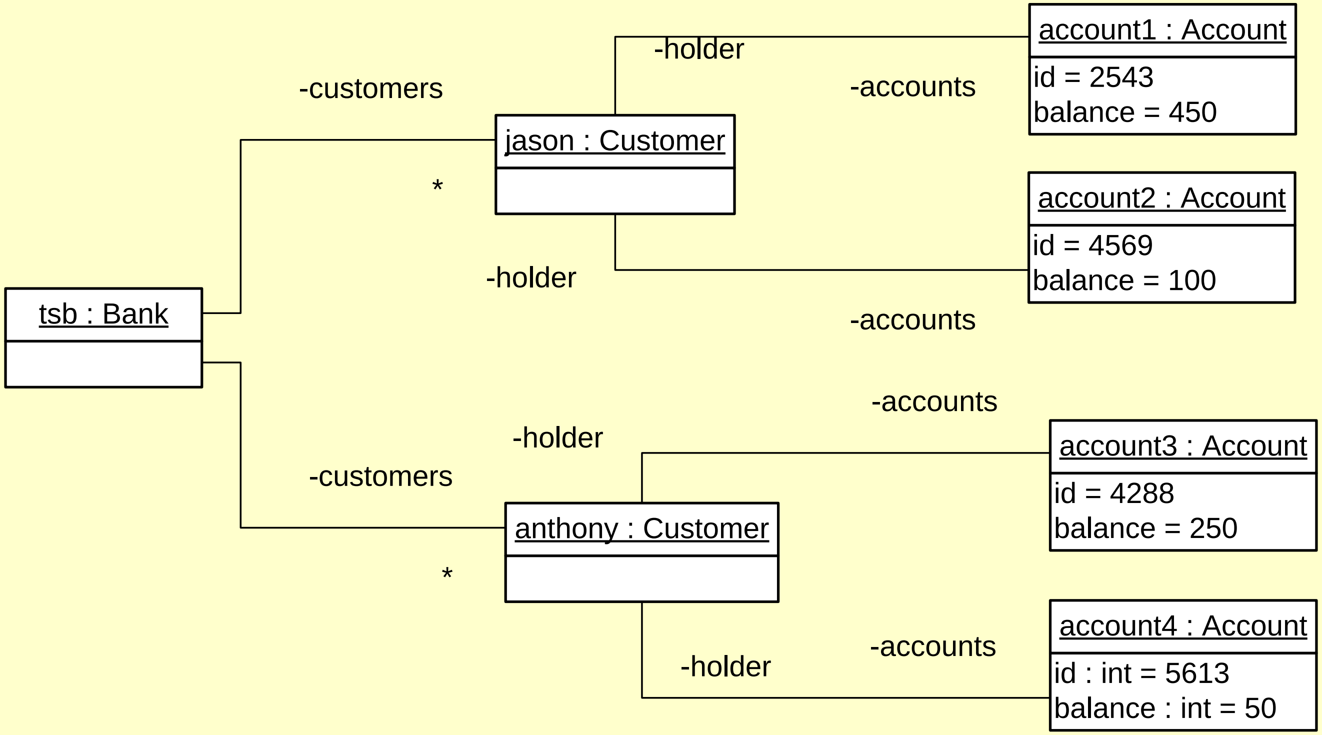
-holder

account3 : Account

id = 4288
balance = 250

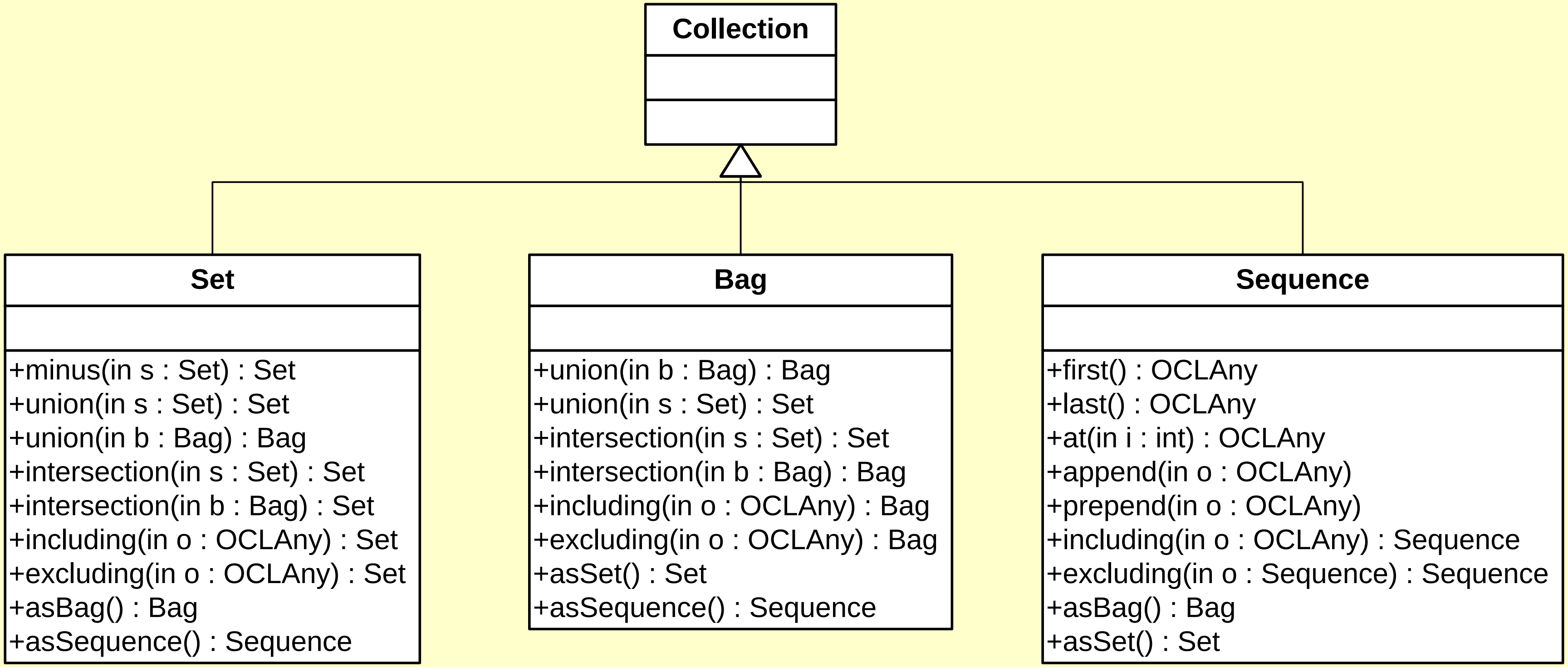
jason.accounts->forAll(a : Account | a.balance > 0) = true
jason.accounts->select(balance > 100) = {account1, account3}
jason.accounts->includes(account4) = true
jason.accounts->exists(a : Account | a.id = 333) = false
jason.accounts->includesAll({account1, account2}) = true
jason.accounts.balance->sum() = 850
jason.accounts->collect(balance) = {450, 100, 250, 50}

Navigating across and Flattening Collections



tsb.customers.accounts = {account1, account2, account3, account4}
tsb.customers.accounts.balance = {450, 100, 250, 50}

Specialized Collection Operations



Collection

Set

+minus(in s : Set) : Set
+union(in s : Set) : Set
+union(in b : Bag) : Bag
+intersection(in s : Set) : Set
+intersection(in b : Bag) : Set
+including(in o : OCLAny) : Set
+excluding(in o : OCLAny) : Set
+asBag() : Bag
+asSequence() : Sequence

Bag

+union(in b : Bag) : Bag
+union(in s : Set) : Set
+intersection(in s : Set) : Set
+intersection(in b : Bag) : Bag
+including(in o : OCLAny) : Bag
+excluding(in o : OCLAny) : Bag
+asSet() : Set
+asSequence() : Sequence

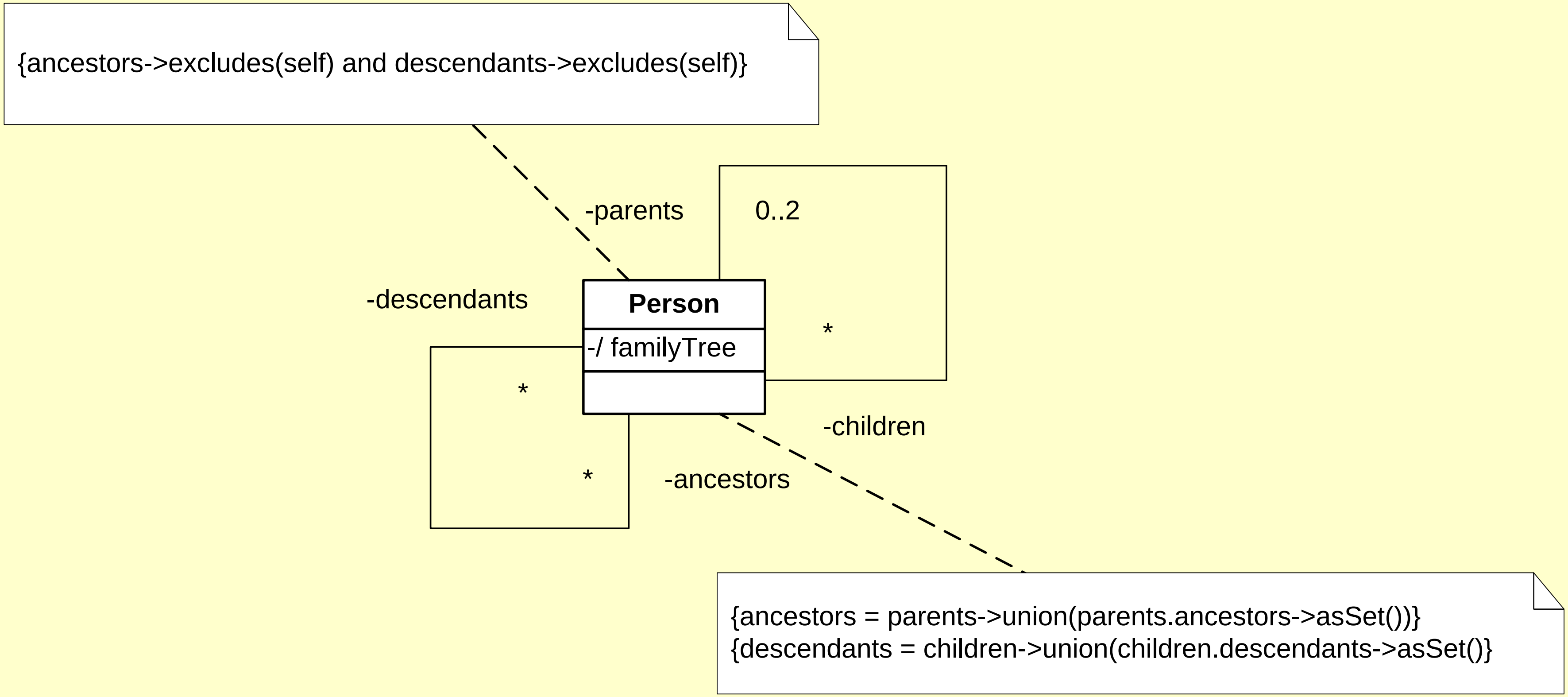
Sequence

+first() : OCLAny
+last() : OCLAny
+at(in i : int) : OCLAny
+append(in o : OCLAny)
+prepend(in o : OCLAny)
+including(in o : OCLAny) : Sequence
+excluding(in o : Sequence) : Sequence
+asBag() : Bag
+asSet() : Set

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}

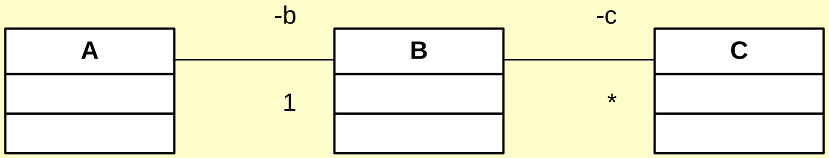
Set Operations: Example



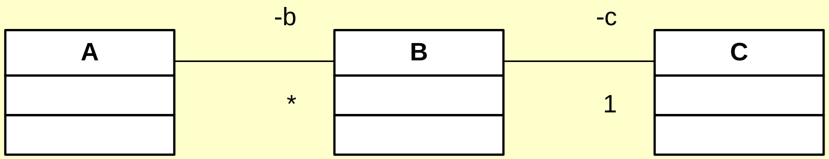
{ancestors->excludes(self) and descendants->excludes(self)}

{ancestors = parents->union(parents.ancestors->asSet())}
{descendants = children->union(children.descendants->asSet()}

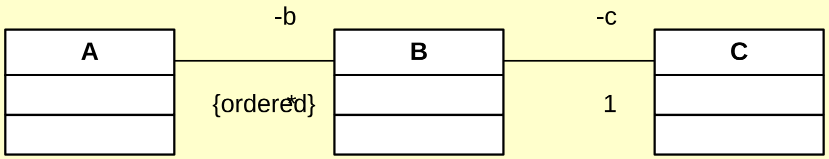
Navigability & Collections



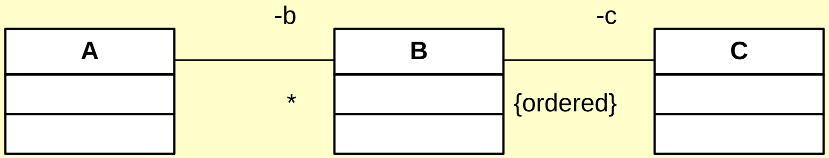
```
Context A :
self.b.c : Set(C)
```



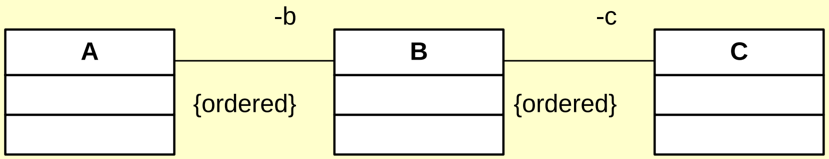
```
self.b.c : Bag(C)
```



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



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



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

Iterators : examples

- `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 p inv:
       p.allInstances() -> forAll(p1, p2 |
              p1 <> p2 implies p1.nom <> p2.nom)
  ```

Conditional Constraints

- Constraints which depend from other constraints
- Can be expressed in two ways:
  a. if expr1 then expr2 else expr3 endif: if expr1 is true then expr2 must be true, otherwise expr3 must be true
  b. expr1 implies expr2: if expr1 is true, then expr2 must be true. If expr1 is false, then the whole expression is true

Conditional Constraints: Examples

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

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

Variables

- Variables can be used to improve readability of complex constraints
- OCL syntax: `let … in …`

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

- To make it accessible from anywhere: `def`

```
context Personne
def: argent : int = compte.solde -> sum()

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