OCL
Expression Language

Basic Types, Enumerated Types,

Types Constructors (Set, Bag, Sequence, OrderedSet, Tuples)

Expressions (op, ., -> if, let)

Operations on collections ->

Iterators (on collections) →

Operations on types

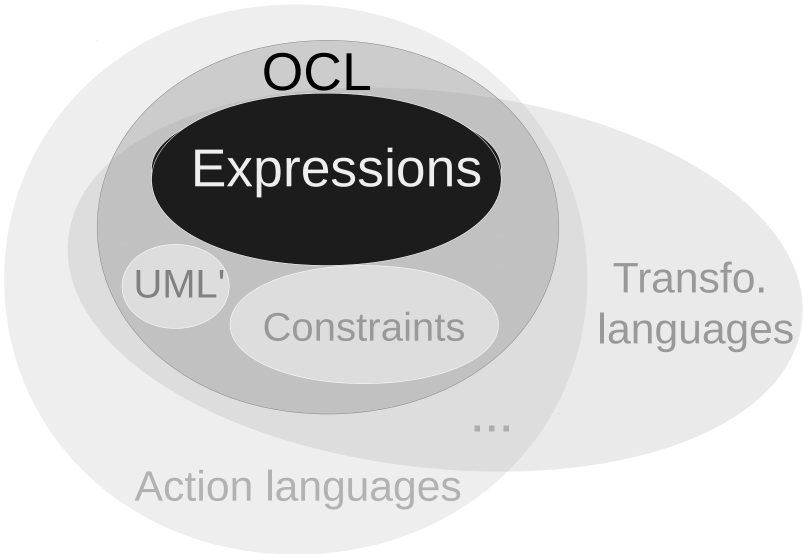
OCL
The Expression Language

- Subset of OCL

- **No assignment**
- No explicit iteration
- +/- functional language (if, let)

- Can serve as a **query language**

- Almost independent from UML
- **Integrated** into other languages



Expressions

UML-based expressions

UML-free expressions

Syntax of expression

UML-free expressions

(age<40 **implies** salary>1000) **and** (age>=40 **implies** salary>2000)

salary > (**if** age<40 **then** 1000 **else** 2000 **endif**)

let m:Integer = **Set**{2,3}->**max**() **in**
    Sequence{1..m}->**collect**(i|i*i)->**append**(999)

Set{2,3}->**product**(**Set**{'a','b'})->**includes**(**Tuple**{*first*=2,*second*='b'})

Sequence{2,5,3}**->collect**(i|i*i+1) = **Sequence**{5,26,10}

UML-based Expressions

self.*salary* **–** *100*

self.*children***->isEmpty***()*

self.*children***->forall***(age>20)*

self.*getTaxes(1998) /* **self.***children***->size***()*

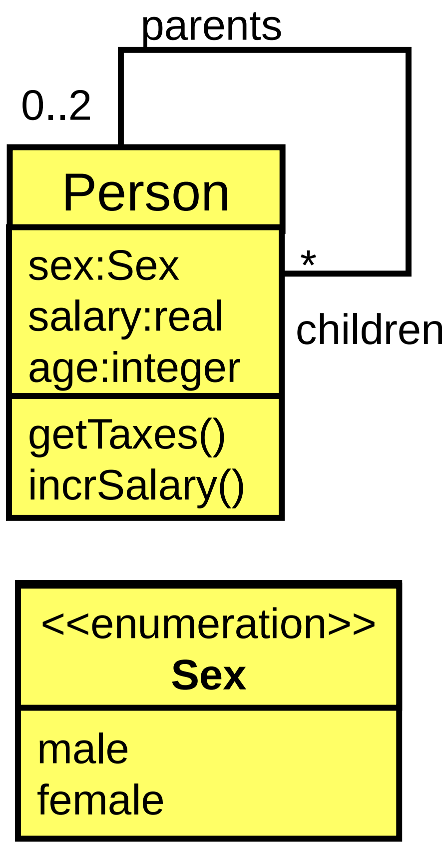
self.*children***->select***(sex= Sex::male)*

self.*children***->collect(***salary)***->sum***()*

self.children.*salary***->sum***()*

self.*children***->union***(***self**.*parents)***->collect***(age)*

self.salary > **self**.salary@**pre**



Syntax of Expressions
(simplified)

<const>	12	'hello'
<id>	children	Residence
<expr> <op> <expr>	age>18	
self	« **self** » in python, « **this »** in java	
<exprobj> **.** <prop>	Tuple{x=2,y=3}.x	self.x
<exprobj> **.** <objprop>**(**<expr>...**)**	self.chairman.multiplySalary(10)	
<exprcoll> **->** <collprop>**(**<expr>...**)**	self.employees->collect(salary)	
<package>**::**<package> … **::** <element>	like in C++, « . » in python and java	
if <expr> **then** <expr> **else** <expr> **endif**	if age>18 then 1200 else 50*age endif	
let <id> **:** <type> **in** <expr> **.**<expr>	let e=x*x+y in (e-20)*1.5	

Types and Values

Basic types

Enumerations

Collections

Tuples

Types

- Basic types
  - String
  - Integer
  - Real
  - Boolean
- Collections
  - Set(T)
  - Sequence(T)
  - Bag(T)
  - OrderedSet(T)
- Meta-types
  - OclVoid
  - OclAny
  - OclType
  - OclState
  - OclExpression
- Enumerations
- Tuples
  - TupleType(
    x:T1,
    y:T2,
    …)
- + Types from UML
  - Class
  - Association
  - …

Values

- 'hello' : String
- 13 : Integer
- -5.6 : Real
- true : Boolean
- Day::monday : Day -- an enumeration
- **Undefined** : OclVoid
- Set{1,24,1,5,12} : Set(Integer)
- Sequence{1,24,1,5} : Sequence(Integer)
- Bag{1,24,1,5,12} : Bag(Integer)
- OrderedSet{1,1,4,2,6,1} : OrderedSet(Integer)
- Tuple{x=0.5, y='ok'} : TupleType(x:Real,y:String)

Basic types

Integer

Real

Boolean

String

Numbers

- ***Integer***
  - values : *1, -5, 34, 24343*, ...
  - operations : *+, -, *, div, mod, abs, max, min*
- ***Real***
  - values : *1.5, 1.34*, ...
  - operations : *+, -, *, /, floor, round, max, min*

Integer type « conforms » to the type ***Real***

Boolean

- ***Boolean***
  - values : **true**, **false**
  - operations : *not*, *and*, *or*, *xor*, *implies*, *if-then-else-endif*

partial evaluation :

true or *x*	always true, even when x is Undefined
false and *x*	always false, even when x is Undefined

(age<40 **implies** *salaire>1000)* **and** *(age>=40* **implies** *salaire>2000)*

if *age<40* **then** *salaire > 1000* **else** *salaire > 2000* **endif**

*salaire > (***if** *age<40* **then** *1000* **else** *2000* **endif***)*

Strings

Values:

''

'word is a word'

Operations:

=

s.size*()*

*s1.*concat*(s2)*

*s1.*substring*(i1,i2)*

s.toUpperCase*()*

s.toLowerCase*()*

s.characters()

name = nom**.substring**(1,1)**.toUpperCase().concat(**
name**.substring**(2,name**.size()**)**.toLowerCase**())

'word is a word'.characters()->count(' ')

Enumerations

Enumeration

- Values

  Day::Tuesday (previous notation: *#Tuesday*)

- Operators

  =, <>

<<enumeration>>
Day

monday
tuesday
wednesday
thursday
friday
saturday
sunday

No ordering relation

Tuples

Tuples

- Types tuples ( )
  - TupleType(x : real, y : real)
  - TupleType(y : real, x : real)
  - TupleType(nom : string, prénom : string, age : integer)
- Valeur tuples { }
  - Tuple{ x=-1.5 , y=12.6 }
  - Tuple{ y:real=12.6, x:real=-1.5 }
  - Tuple{ nom = 'dupont', prénom='léon', age=43 }
- Operations
  - Tuple{ *x=-1.5 , y=12.6* }.*x*
  - *enfants.adresse.ville*

  See also the cartesian product on collection *->product*

Collections

unique/nonunique
ordered/nonordered
Bags
Sets
OrderedSet
Sequence
. vs. ->

Collections

	{unique}	{nonunique}
{unordered}	**Set(*T*)**	**Bag(*T*)**
{ordered}	**OrderedSet(*T*)**	**Sequence(*T*)**

General Type **Collection(*T*)**

Collections

- **Set**
  - No duplicates, order is not important
  - ***Set*** *{ 1, 5, 10, 3 }* *= Set{1,5,5,1,10,3}*
- **OrderedSet**
  - No duplicates, order is important
  - ***OrdernedSet*** *{ 1, 5, 10, 3 }* *<> OrderedSet{1,10,5,3}*
- **Bag**
  - Possible duplicates, order is not important
  - ***Bag*** *{ 1, 5, 5, 10, 3, 1 }* *= Bag {5,1,10,3,1,5}*
- **Sequence**
  - Possible duplicates, order is important
  - ***Sequence*** *{ 1, 5, 5, 10, 3, 1 }* *<> Sequence {1,5,10,5,3,1}*

Collection Expressions

Set { *'lundi'*, *'mercredi'*, *'mardi'* }

Bag { *'lundi'*, *'lundi'*, *'mardi'*, *'lundi'* }

OrderedSet { *10*, *20*, *5* }

Sequence { *'lundi'*, *'lundi'*, *'mardi'*, *'lundi'* }

To specify ranges

Sequence { *1..5, 2..4* }

Useful for 'loops' :

Sequence { *0* .. *n-1* }->**forall**(*i* | …)

Operations on collections : ->

Set *{ 3, 5, 2, 45, 5 }->***size***()*

Sequence *{ 1, 2, 45, 9, 3, 9 }* **->count***(9)*

Sequence *{ 1, 2, 45, 2, 3, 9 }* **->includes***(45)*

Bag *{ 1, 9, 9, 1 }* **-> count***(9)*

*c->***asSet***()***->size***() = c***->size***()*

*c->***count(***x***)** *= 0*

Bag *{ 1, 9, 0, 1, 2, 9, 1 } ->* **includesAll***(* **Bag***{ 9,1,9})*

IMPORTANT

. vs ->

. to get access to properties of objects

-> to get access to properties of collections

+ some rules to mix collections and objects (see later)

self.*taxes(1998)* / self.*children* ->
size*()*

because self
is an object

because self.children
is a collection of objects

Element vs. Singleton

In almost all languages

≠
- one element 42
- from singleton containing this element Set{42}

Implicit conversion in OCL **element => singleton**
when an collection operation
is applied to an element

elem **->** *prop* ⟺ **Set{***elem***}->***prop*

Operations on collections

Operations on Collections (1/3)

- Set theory
  - *->union*, *->intersection*, ...
- Collection specifics
  - *->at*, *->subSequence*, ...
- Predicate filtering
  - *->select*, *->reject*, *->any*
- Image
  - *->collect*, *->collectNested*
- Universal quantifiers
  - *->forall*, *->exists*, *->one*, *->isUnique*
- General iterator
  - *->iterate*

Operations on Collections (2/3)

Defined on all kind of collections (Bag, Set, OrderedSet, Sequence)

Cardinality:	->**size**()
Emptyness:	->**isEmpty**()
Non emptyness:	->**notEmpty()**
Occurrence number:	->**count**(elem)
Membership:	->**includes**(elem)
Non membership:	->**excludes**(elem)
Inclusion:	->**includesAll**(coll)
Exclusion:	->**excludesAll**(coll)

Operations on Collections (3/3)

Defined on all kind of collections (Bag, Set, OrderedSet, Sequence)

Sum/max/min:	->**sum**()
	->**max**()
	->**min**()
Cartesian Product:	->**product**(coll)
Flatten nested collections:	->**flatten**()
Conversions	->**asBag**()
	->**asSet**()
	->**asOrderedSet**()
	->**asSequence**()

Operations on Sets

Union	ens -> **union**(ens)
Intersection	ens -> **intersection**(ens)
Difference	ens1 – ens2
Symmetric difference	ens -> **symmetricDifference()**
Adding an element	ens -> **including**(elem)
Removing an element	ens -> **excluding**(elem)

Plus operations defined on Collections

Iterators

->select	->reject	->any
->forall	->exist	->one
->unique		
->collect		

Filters
->select ->reject ->any

->**select**(*cond*)

select elements satisfying the condition

->reject(cond)

reject elements

->**any**(*cond*)

select any element satisfying the condition

- non determinist
- useful when there is only one element
- Undefined if the collection is empty

Filters
Examples

Set*{2,4,-1,-23,5,6}->***select***(i:Integer | i<0)*

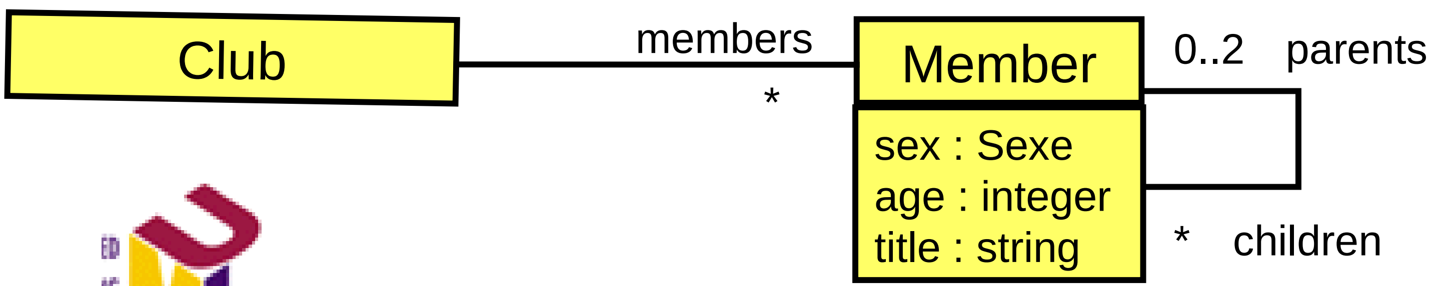
Set*{2,4,-1,-23,5,6}->***reject***(i:Integer | i>=0)*

Set*{2,5,9,0,-1,8}->***any***(i : i<0)*

self*.children ->***select***(age>10* **and** *sex = Sex::Male)*

self*.children ->***reject***(e : Person | e.children->***isEmpty**()*)->***notEmpty**()

*members->***any***(title='president')*

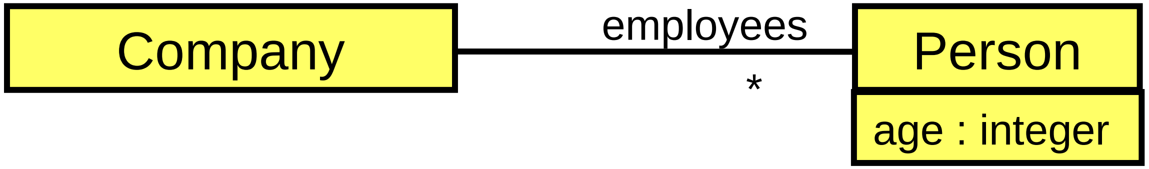


Iterators
Alternative syntaxes

self.*employees->***select***(age > 50)*

self.*employees->***select***(p | p.age>50)*

self.*employees->***select***(p : Person | p.age>50)*



Quantifiers
->forall ->exists ->one

$\forall$ $\exists$ $\exists!$

Set{1,8,9,5,-1,6,-5}->**forall**(*n*| *n<10)*

Sequence{1,8,9,5,-1,6,-5}->**exists**(*n*| *n<0)*

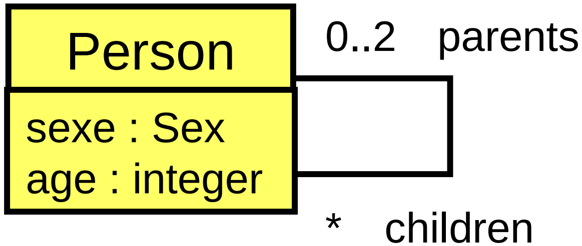
not Sequence{1,8,9,5,-1,6,-5}->**one**(*n*| *n>0)*

Sequence{1,8,9,5,-1,6,-5}->**one**(*n*| *n<0)*

self.*enfants*->**forall**(*age<10)*

self.*enfants*->**exists**(*sexe=Sexe::Masculin)*

self.*enfants*->**one**(*age>=18)*



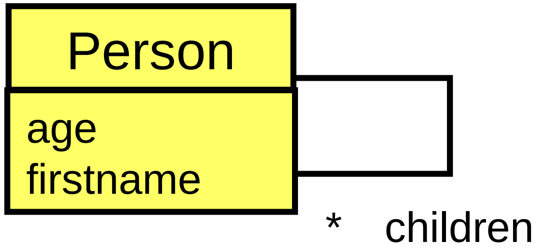
Syntax Comparison

$\forall$ $\exists$ $\exists!$

$\forall\ c \in children\ .\ \ c.age < 10$

*children->***forall***(p : Person | p.age<10)*

Quantifiers
Alternative syntaxes



It's possible to

- give a name to a variable
- explicit its type
- use various variables at the same time

```
self.children->forall( age < self.age )
self.enfants->forall( e | e.age < self.age - 7)
self.children->forall( e : Person | e.age < self.age - 7)
self.children->exists( e1,e2 | e1.age = e2.age )
self.children->exists( e1,e2 | e1.age = e2.age and e1<>e2 )
self.children->forall( e1,e2 : Person |
                          e1 <> e2 implies e1.firstname <> e2.firstname)
```

->isUnique

coll -> **isUnique** *(expr)*

True if all elements return a different value for expr

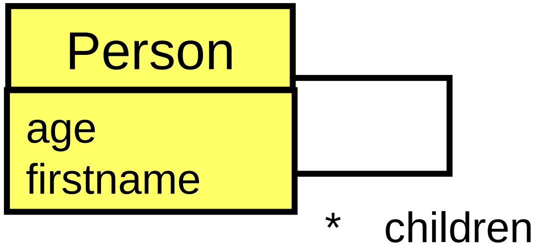
not Set*{1,2,-2,3}->***isUnique***(n|n*n)*

- **self***.children ->* **isUnique** *(firstname)*

instead of

self*.children->***forall***(p1,p2 : Person |*

p1 <> p2 implies p1.firstname <> e2.firstname)



Useful to define the notion of "imported key" for instance

->isUnique vs. {unique}

self.*children* -> **isUnique** *(firstname)*

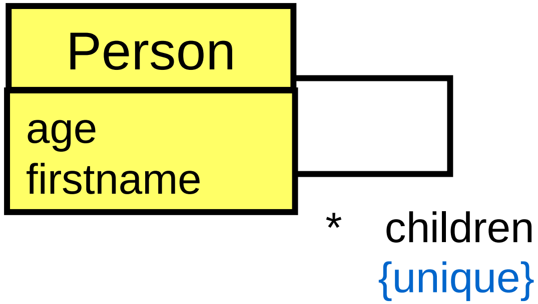


Image of an expression ->collect

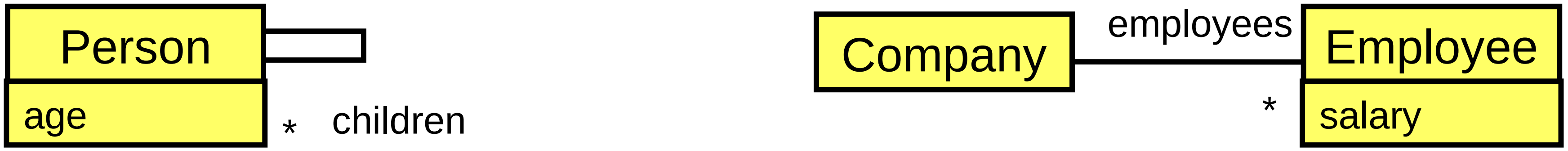
coll -> **collect***(expr)*

- "image" of a function (map, apply, ...)
- expr evaluated for each element
- result in:
  - a **Bag** if *coll* is a Set or a Bag
  - a Sequence if coll is a Sequence or a OrderedSet

Set{1,2,-1}->**collect**(i|i*i) = Bag{1,4,1}

self.children-**>collect***(age) = Bag{10,5,10,7}*

self.*employees***->collect(***salary/10)*->**sum**()

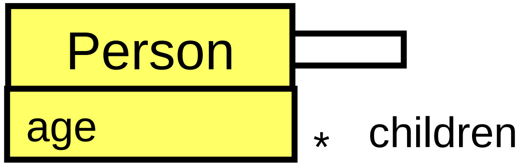


->collect
Simplified Syntax

***self**.children.age*

***self**.children->**collect**(age)*

. with a collection ↔ **collect**



Sorting
->sortedBy

- **Set**{1,4,0,2,-3}->**sortedBy**(n | n*n)
- **self**.children->**sortedBy**(age)->**last**()
- **self**.children

  ->**collect**(e1,e2 | e1.age-e2.age)

  ->**select**(d | d>0)

  ->**sortedBy**(e | e)->**last**()

Operations with types

oclIsTypeOf

oclIsKindOf

oclAsType

oclIsTypeOf, oclIsKindOf

- 18.**oclIsTypeOf**(Integer) true
- 18.5.**oclIsTypeOf**(Integer) false
- 18.**oclIsKindOf**(Real) true

- **Set**{1, 0.5,1.5}->**select**(**oclIsTypeOf**(Real))

  Set{0.5, 1.5} : Set(Real)

- **Set**{1, 0.5,1.5}->**select**(**oclIsKindOf**(Real))

  Set{1, 0.5, 1.5} : Set(Real)