```
use> ------------------------------------------------------------------------
use>
use> -- Operations on all collection kinds (Set, Bag, Seq, Ord)
use> -- ------------------------------------------------
use> --
use> -- Set{...}, Bag{...}, Sequence{...}, OrderedSet{...}
use> -- oclEmpty(Col(Type))), e.g., oclEmpty(Set(String)), ...
use> -- Set{}, Sequence{}, ...
use> --
use> -- including, excluding (Col Elem -> Col)
use> -- includes, excludes (Col Elem -> Boolean)
use> -- size (Col -> Integer), isEmpty, notEmpty (Col -> Boolean)
use> --
use> -- select (Set Pred -> Set, Bag Pred -> Bag, Seq Pred -> Seq),
use> -- collect (Set Expr -> Bag, Bag Expr -> Bag, Seq Expr -> Seq)
use> --
use> -- forAll, exists (Col Pred -> Boolean)
use> -- iterate (Col Expr -> ResultType)
use> --
use> -- union (Col Col -> Col)
use> --
use> -- reject (as select)
use> -- one, isUnique (as exists)
use> -- sortedBy (Col Expr-On-Basic-Data-Type -> Seq)
use> --
```

```
use> -- Operations on special collections
use> -- --------------------------------
use> --
use> -- first, last, at, subsequence (on Seq)
use> -- sum (on Col(Integer), Col(Real))
use> -- ...
use> --
use> -- Further operations see OCL Standard Library

use> ------------------------------------------------------------------

use> ?Set{11,22,33}=Set{22,33,11}
     true : Boolean

use> ?Set{11,22,33}=Set{11,22,11,33,22,22,11}
     true : Boolean

use> ?Bag{11,11,22,33}=Bag{22,33,11}
     false : Boolean

use> ?Sequence{11,22}=Sequence{22,11}
     false : Boolean

use> ?oclEmpty(Set(Integer))->including(11)->including(22)->including(22)
     Set{11,22} : Set(Integer)
```

```
use> ?Set{11,22,33}->excluding(11)
     Set{22,33} : Set(Integer)

use> ?Set{11,22,33}->excluding(44)
     Set{11,22,33} : Set(Integer)

use> ?Bag{11,22,11,33}->excluding(11)
     Bag{22,33} : Bag(Integer)

use> ?Sequence{11,22,11,33}->excluding(11)
     Sequence{22,33} : Sequence(Integer)

use> -- Col: Set, Bag, Seq
use> --
use> -- Commutativity (Kommuntativitaet):
use> -- C->including(E1)->including(E2) = C->including(E2)->including(E1)
use> --
use> -- Absorption (Absorption):
use> -- C->includes(E) implies C->including(E) = C
use> --
use> --                Set Bag Seq Ord
use> -- Commutativity +   +   -   -
use> -- Absorption    +   -   -   +
```

```
use> ------------------------------------------------------------------------

use> -- Set{1,2,2,1} = Set{1,1,2,2} = Set{1,2} = Set{2,1}
use> --
use> -- Bag{1,2,2,1} = Bag{1,1,2,2} <> Bag{1,2} = Bag{2,1}
use> --
use> -- Seq{1,2,2,1} <> Seq{1,1,2,2} <> Seq{1,2} <> Seq{2,1} pairwise distinct
use> --
use> -- Bag{1,2,2,1}->asSet() = Set{1,1,2,2}
use> --
use> -- Seq{1,2,2,1}->asBag() = Bag{1,2,2,1}

use> ------------------------------------------------------------------------

use> ?oclEmpty(Set(Integer))->including(22)->including(11)->excluding(33)
     Set{11,22} : Set(Integer)

use> ?Set{11,22}->includes(22) and Set{11,22}->excludes(33)
     true : Boolean

use> ------------------------------------------------------------------------
```

```
use> ?Set{-2,-1,0,1,2}->select(i|i>=0)
     Set{0,1,2} : Set(Integer)

use> ?Set{-2,-1,0,1,2}->collect(i|i*i)
     Bag{0,1,1,4,4} : Bag(Integer)

use> ?Set{-2,-1,0,1,2}->collect(i|if i>=0 then true else false endif)
     Bag{false,false,true,true,true} : Bag(Boolean)

use> ----------------------------------------------------------------

use> ?Set{-2,-1,0,1,2}->forAll(i|i.mod(2)=0)
     false : Boolean

use> ?Set{-2,-1,0,1,2}->exists(i|i.mod(2)=0)
     true : Boolean

use> ?Set{-2,-1,0,1,2}->iterate(i:Integer;res:Boolean=false|res or i.mod(2)=0)
     true : Boolean
```

```
use> ?Set{-2,-1,0,1,2}->iterate(i:Integer;
       res:Set(Sequence(OclAny))=oclEmpty(Set(Sequence(OclAny)))|
         res->including(Sequence{i,
                        i.mod(2),
                        if i.mod(2)=0 then 'even' else 'odd' endif}))

     Set{Sequence{-2,0,'even'},
         Sequence{-1,-1,'odd'},
         Sequence{0,0,'even'},
         Sequence{1,1,'odd'},
         Sequence{2,0,'even'}} : Set(Sequence(OclAny))

use> ----------------------------------------------------------------

use> ?Set{11,22,33}->union(Set{33,44})
     Set{11,22,33,44} : Set(Integer)

use> ?Set{-2,-1,0,1,2}->reject(i|i<0)
     Set{0,1,2} : Set(Integer)

use> ?Set{-2,-1,0,1,2}->one(i|i>0 and i.mod(2)=0)
     true : Boolean
```

```
use> ?Set{-2,-1,0,1,2}->isUnique(i|i)
     true : Boolean

use> ?Set{-2,-1,0,1,2}->isUnique(i|i*i)
     false : Boolean

use> ?Set{-2,-1,0,1,2}->isUnique(i|i*i*i)
     true : Boolean

use> ?Set{Sequence{22,'C'},Sequence{11,'B'},Sequence{33,'A'}}->
     sortedBy(e|e->first()->oclAsType(Integer))
     Sequence{Sequence{11,'B'},
              Sequence{22,'C'},
              Sequence{33,'A'}} : Sequence(Sequence(OclAny))

use> ?Set{Sequence{22,'C'},Sequence{11,'B'},Sequence{33,'A'}}->
     sortedBy(e|e->at(2)->oclAsType(String))
     Sequence{Sequence{33,'A'},
              Sequence{11,'B'},
              Sequence{22,'C'}} : Sequence(Sequence(OclAny))

use> ------------------------------------------------------------
```

Collection operation iterate

- COLEXPR->iterate(ELEMVAR:ELEMTYPE; RESVAR:RESTYPE=INITEXPR | ITEREXPR)

- COLEXPR, INITEXPR, ITEREXPR: OCL expression
  ELEMVAR, RESVAR: OCL variables
  ELEMTYPE, RESTYPE: OCL types

  type(COLEXPR) in
    {Set(ELEMTYPE), Bag(ELEMTYPE), Sequence(ELEMTYPE), OrderedSet(ELEMTYPE)}

  type(INITEXPR) = *type*(ITEREXPR) = RESTYPE

- Also allowed: COLEXPR->iterate(ELEMVAR; RESVAR:RESTYPE=INITEXPR | ITEREXPR)
  i.e., ':ELEMTYPE' is optional

- Collection operations can be expressed with iterate

- Example

  ibm.worker->exists(p:Person | p.fName='Bob')

  ibm.worker->iterate(p:Person; bobEx:Boolean=false | bobEx or p.fName='Bob')

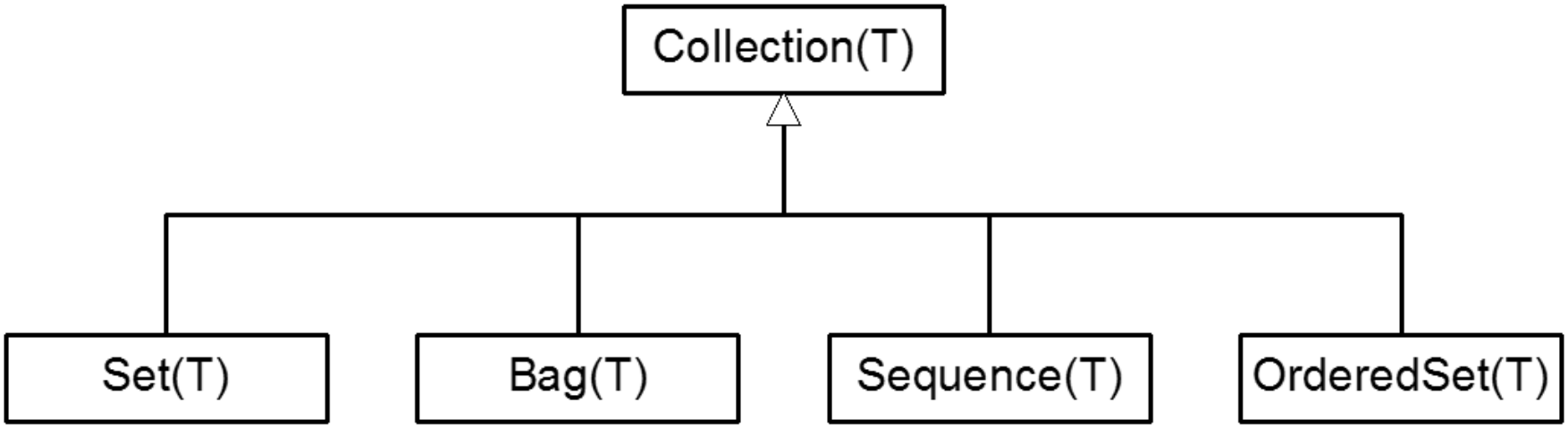
  COLEXPR ibm.worker
  ELEMVAR p
  ELEMTYPE Person
  RESVAR bobEx
  RESTYPE Boolean
  INITEXPR false ibm.worker = Set{ada,bob} ->
  ITEREXPR bobEx or p.fName='Bob' false or ada.fName='Bob' or bob.fName='Bob'

- iterate Evaluation in Java-like Pseudo Code

```
COLEXPR->iterate(ELEMVAR:ELEMTYPE; RESVAR:RESTYPE=INITEXPR | ITEREXPR)

RESTYPE iterate() {
  ELEMTYPE ELEMVAR;
  RESTYPE RESVAR = INITEXPR;
  for (Iterator i = COLEXPR.iterator(); i.hasNext();) {
    ELEMVAR = (ELEMTYPE)i.next();
    RESVAR = ITEREXPR;
  };
  return RESVAR;
}
```

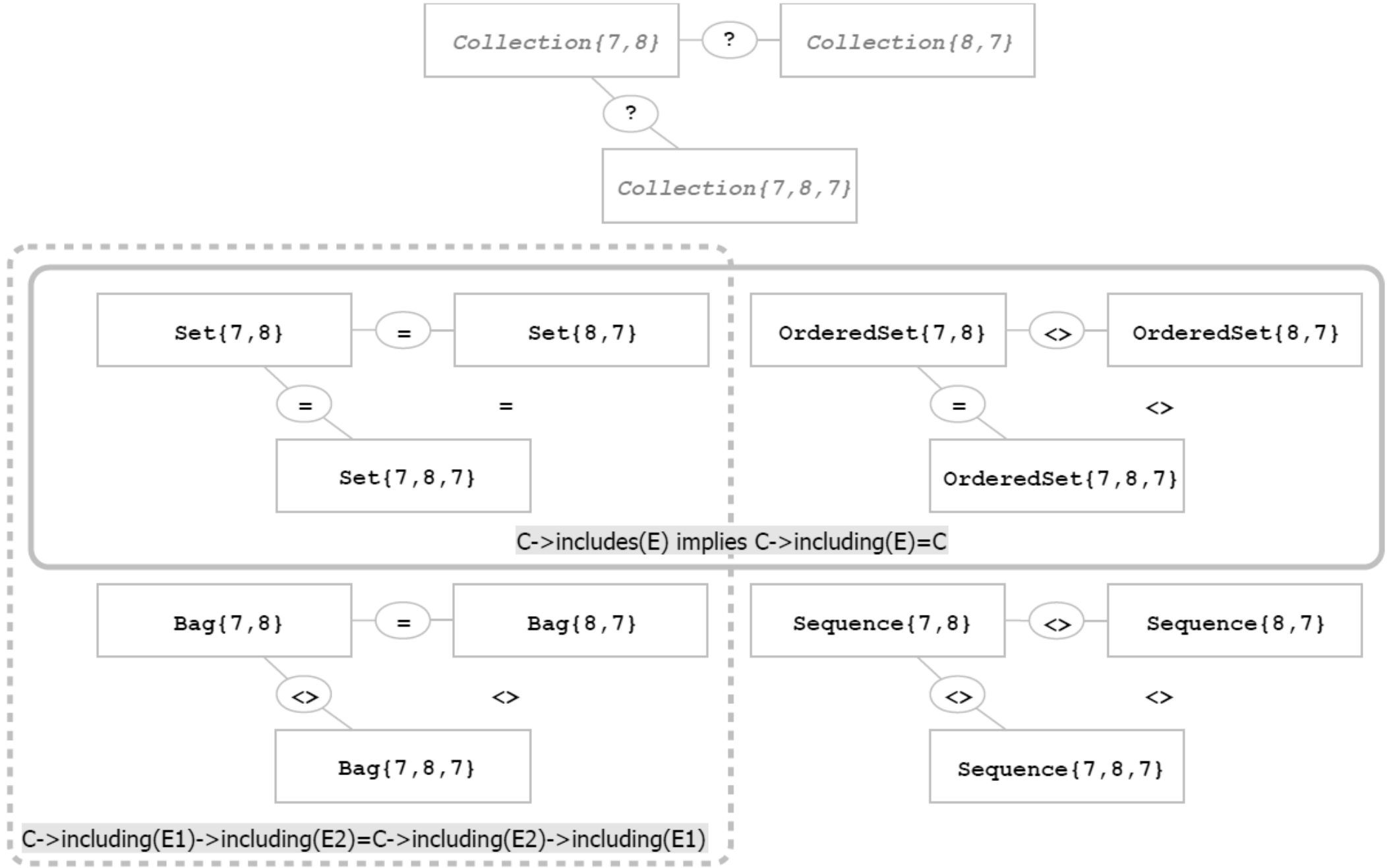
Set versus Bag versus Sequence versus OrderedSet



CountingOne	+	-	-	+
Exchanging	+	+	-	-

CountingOne `COL->forAll(e|COL->count(e)=1)`
Exchanging `COL->incl(X)->incl(Y)=COL->incl(Y)->incl(X)`

CountingOne Collection elements appear in the collection only once
Exchanging Insertion order of collection elements does not matter


Collection{7,8}
?
Collection{8,7}
?
Collection{7,8,7}
Set{7,8}
=
Set{8,7}
=
=
Set{7,8,7}
OrderedSet{7,8}
<>
OrderedSet{8,7}
=
<>
OrderedSet{7,8,7}
C->includes(E) implies C->including(E)=C
Bag{7,8}
=
Bag{8,7}
<>
<>
Bag{7,8,7}
Sequence{7,8}
<>
Sequence{8,7}
<>
<>
Sequence{7,8,7}
C->including(E1)->including(E2)=C->including(E2)->including(E1)

collection kinds
- Set
- Bag
- Sequence
- OrderedSet

collection operations
- Set, Bag, Sequence, OrderedSet
- =, <>
- including, excluding
- includes, excludes, includesAll, excludesAll
- isEmpty, notEmpty, size
- select, reject
- collect, collectNested
- forAll, exists
- one, any, isUnique, sortedBy, union
- iterate
- asSet, asBag, asSequence, asOrderedSet
- flatten
- allInstances
- oclEmpty

```
- Set{7,8,9} Set{8,7,9,7}
  Bag{7,8,9} Bag{8,7,9,7}
  Sequence{7,8,9} Sequence{8,7,9,7}
  OrderedSet{7,8,9} OrderedSet{8,7,9,7}
- (Set{7,8,9}=Set{8,7,9,7})=true
  (Bag{7,8,9}<>Bag{8,7,9,7})=true
  (Sequence{7,8,9}<>Sequence{8,7,9,7})=true
  (OrderedSet{7,8,9}<>OrderedSet{8,7,9,7})=true
  (Bag{8,7,9,7}=Bag{7,7,8,9})=true
  (OrderedSet{8,7,9}=OrderedSet{8,7,9,7})=true
- Set{}->including(7)->including(8)->including(9)
  Set{7,8,9}->including(6)->excluding(6)->excluding(5)
- Set{7,8,9}->includes(8)=true
  Set{7,8,9}->includes(5)=false
  Set{7,8,9}->excludes(8)=false
  Set{7,8,9}->excludes(5)=true
  Set{7,8,9}->includesAll(Set{7,9})=true
  Set{7,8,9}->includesAll(Set{6,7})=false
  Set{7,8,9}->includesAll(Set{5,6})=false
  Set{7,8,9}->excludesAll(Set{7,9})=false
  Set{7,8,9}->excludesAll(Set{6,7})=false
  Set{7,8,9}->excludesAll(Set{5,6})=true
```

```
- Set{7}->excluding(7)->isEmpty()=true
  Set{7}->notEmpty()=true
  Set{7,8,9}->size()=3
- Set{7,8,9}->select(i|i.mod(2)=1)=Set{7,9}
  Set{7,8,9}->reject(i|i.mod(2)=1)=Set{8}
- Set{7,8,9}->collect(i|i*i)=Bag{49,64,91}: Bag(Integer)
  Set{7,8,9}->collect(i|if i.mod(2)=0
    then 'Even' else 'Odd' endif)=
    Bag{'Even','Odd','Odd'}: Bag(String)
- Set{7,8}->collectNested(i|Sequence{i,i*i})=
  Bag{Sequence{7,49},Sequence{8,64,}}: Bag(Sequence(Integer))
- Set{7,8,9}->forAll(i|0<=i and i<=9)=true
  Set{7,8,9}->forAll(i|0<i and i<9)=false
  Set{7,8,9}->exists(i|i.mod(2)=0)=true
  Set{7,8,9}->exists(i|i.mod(5)=0)=false
```

```
- Set{7,8,9}->one(i|i.mod(2)=0)=true
  Set{7,8,9}->one(i|i.mod(2)=1)=false
  Set{7,8,9}->any(true)=7
  Set{7,8,9}->any(i|i.mod(2)=0)=8
  Set{7,8,9}->isUnique(i|i*i)=true
  Set{7,8,9}->isUnique(i|i.mod(2)=0)=false
  Set{7,8,9}->sortedBy(i|-i)=Sequence{9,8,7}
  Set{7,8,9}->sortedBy(i|if i.mod(2)=0
    then 'E' else 'O' endif)=
    Sequence{8,7,9}
  Sequence{-8,9,-7}->sortedBy(i|i.abs)=Sequence{-7,-8,9}
  Set{7,8}->union(Set{9,8})=Set{7,8,9}
  Bag{7,8}->union(Bag{9,8})=Bag{7,8,8,9}
  Sequence{7,8}->union(Sequence{9,8})=Sequence{7,8,9,8}
  OrderedSet{7,8}->union(OrderedSet{9,8})=OrderedSet{7,8,9}

other OCL keywords [not directly related to collections]
- null, oclUndefined, isDefined, isUndefined
- oclIsTypeOf, oclIsKindOf, oclAsType
- let
- if then else endif
```