OCL operation reference

OCL operation reference

Ocl operations for type *Classifier*

allInstances () : Set{T}

Returns a Set containing all of the existing instances of the current classifier (along with instances of all its inherited classifiers).

Expression	Result
let a : String = 'a', b : String = 'b', c : Integer = 2 in String.allInstances()	Set{'a','b'}

Ocl operations for type *OclAny*

oclAsType (typespec : Classifier) : T

Returns *self* statically typed as typespec if it is an instance of this type. *Note* that this does not alter the runtime value of *self*, it only enables access to subtype operations. This operation allows users to cast *self* to another type.

Expression	Result
aPerson.oclAsType(Employee)	an object of Employee type

oclIsInvalid () : Boolean

Returns **true** if *self* is equal to *invalid*.

Expression	Result
let anObject : String = null in anObject.oclIsInvalid()	false
let anObject : String = invalid in anObject.oclIsInvalid()	true
let anObject : String = 'null' in anObject.oclIsInvalid()	false

oclIsKindOf(Classifier typespec) : Boolean

Returns **true** if the type of *self* corresponds to the type or supertype of typespec, **false** otherwise. This operation allows users to check the class hierarchy of *self* much like would an **instanceof** Java.

Expression	Result
anEmployee.oclIsKindOf(Employee)	true
anEmployee.oclIsKindOf(Person)	true
aCat.oclIsKindOf(Person)	false

oclIsTypeOf(typespec : Classifier) : Boolean

Returns **true** if the type of *self* is the same as typespec, or **false** otherwise. This operation allows users to check the exact class type of *self*.

Expression	Result
anEmployee.oclIsTypeOf(Employee)	true
anEmployee.oclIsTypeOf(Person)	false
aCat.oclIsTypeOf(Person)	false

oclIsUndefined () : Boolean

Returns **true** if *self* is equal to **invalid** or **null**.

Expression	Result
let anObject : String = null in anObject.oclIsUndefined()	true
let anObject : String = invalid in anObject.oclIsUndefined()	true
let anObject : String = 'null' in anObject.oclIsUndefined()	false

<> (object : OclAny) : Boolean

Returns **true** if *self* is a different object from *object*.

Expression	Result
let a : String = 'a', b : String = 'a' in a <> b	false
let a : Integer = 2, b : Real = 2.0 in a <> b	false
let a : Integer = -2, b : Integer = 2 in a <> b	true

= (object : OclAny) : Boolean

Returns **true** if *self* is equal to *object*.

Expression	Result
let a : String = 'a', b : String = 'a' in a = b	true
let a : Integer = 2, b : Real = 2.0 in a = b	true
let a : Integer = -2, b : Integer = 2 in a = b	false

< (object : T) : Boolean

Returns **true** if *self* is comparable to *object* and less than *object*.

Expression	Result
let a : Integer = 1, b : Integer = 2 in a < b	true
let a : Real = 1.5, b : Integer = 2 in a < b	true
let a : String = 'Anteater', b : String = 'Aardvark' in a < b	false

> (object : T) : Boolean

Returns **true** if *self* is comparable to *object* and greater than *object*.

Expression	Result
let a : Integer = 1, b : Integer = 2 in a > b	false
let a : Real = 1.5, b : Integer = 2 in a > b	false
let a : String = 'Anteater', b : String = 'Aardvark' in a > b	true

<= (object : T) : Boolean

Returns **true** if *self* is comparable to *object* and less than or equal to *object*.

Expression	Result
let a : Integer = 1, b : Integer = 2 in a <= b	true
let a : Real = 1.5, b : Integer = 2 in a <= b	true
let a : String = 'Anteater', b : String = 'Aardvark' in a <= b	false

>= (object : T) : Boolean

Returns **true** if *self* is comparable to *object* and greater than or equal to *object*.

Expression	Result
let a : Integer = 1, b : Integer = 2 in a >= b	false
let a : Real = 1.5, b : Integer = 2 in a >= b	false
let a : String = 'Anteater', b : String = 'Aardvark' in a >= b	true

Ocl operations for type *String*

-
  - A note on Strings** : OCL Strings begin at index *1*, not *0* as in most languages. Thus *'test'.at(0)* fails in
- invalid* whereas *'test'.at(1)* yields *'t'*. Likewise, *'test'.substring(2, 2)* returns *'e'*.

concat (s : String) : String

Returns a string containing *self* followed by *s*.

Expression	Result
'concat'.concat(' ').concat('operation')	'concat operation'

size () : Integer

Returns the number of characters composing *self*.

Expression	Result
'size operation'.size()	14

substring (lower : Integer, upper : Integer) : String

Returns a string containing all characters from *self* starting from index *lower* up to index *upper* included. Both *lower* and *upper* parameters should be contained between *1* and *self.size()* included. *lower* cannot be greater than *upper*.

Expression	Result
'substring operation'.substring(11, 19)	'operation'
'substring operation'.substring(1, 1)	's'
'substring operation'.substring(0, 1)	***invalid***

toInteger () : Integer

Returns an Integer of value equal to *self*, or |invalid| if *self* does not represent an integer.

Expression	Result
'3.0'.toInteger()	***invalid***
'4'.toInteger()	4
'toInteger'.toInteger()	***invalid***

toLower () : String

Returns *self* with all characters converted to lowercase.

Expression	Result
'LoWeR OpErAtIoN'.toLower()	'lower operation'

toReal () : Real

Returns a Real of value equal to *self*, or |invalid| if *self* does not represent a real.

Expression	Result
'3.0'.toReal()	3.0
'4'.toReal()	4.0
'toReal'.toReal()	***invalid***

toUpper () : String

Returns *self* with all characters converted to uppercase.

Expression	Result
'UpPeR OpErAtIoN'.toUpper()	'UPPER OPERATION'

Ocl operations for type *Number*

In addition to the basic math functions (+, -, /, *) are a number of advanced functions. Take note that *Number* denotes both *Integer* and *Real*, and they're substitutive unless otherwise specified.

Number::abs () : Number

Returns the absolute value of *self*, *self* if it is already a positive number.

Expression	Result
(-2.3).abs()	2.3
-5.abs()	5

Number::floor () : Integer

Returns the integer part of *self* if it is a Real, *self* if it is an Integer.

Expression	Result
(2.3).floor()	2
(2.8).floor()	2
2.floor()	2

Number::max (r : Number) : Number

Returns the greatest number between *self* and *r*.

Expression	Result
6.max(3)	6
6.max(5.2)	6.0
(2.3).max(3)	3.0
(2.3).max(5.2)	5.2

Number::min (r : Number) : Number

Returns the lowest number between *self* and *r*.

Expression	Result
6.min(3)	3
6.min(5.2)	5.2
(2.3).min(3)	2.3
(2.3).min(5.2)	2.3

Number::round () : Integer

Returns the nearest integer to *self* if it is a Real, *self* if it is an Integer.

Expression	Result
(2.3).round()	2
(2.5).round()	3
(2.8).round()	3
2.round()	2

Integer::div (i : Integer) : Integer

Returns the integer quotient of the division of *self* by *i*.

Expression	Result
3.div(2)	1
11.div(3)	3

Integer::mod (i : Integer) : Integer

Returns the integer remainder of the division of *self* by *i*.

Expression	Result
3.mod(2)	1
11.mod(3)	2

Ocl operations for type *Collection*

Please note that OCL collections can contain the *null* value (null) but not the *invalid* value (|invalid|). Trying to add |invalid| within a new or existing collection will yield |invalid| as a result. OCL proposes four distinct kinds of collections offering all possibilities of ordering/unicity.

Collection type	Ordered	Unique
Sequence	true	false
OrderedSet	true	true
Bag	false	false
Set	false	true

any (expr : OclExpression) : T

Returns any element contained in *self* that validates the condition *expr*, null otherwise. Evaluation is shortcut as soon as an element validating *expr* is found. Note that the result of this on unordered collections will be random if more than one element validates *expr*.

Expression	Result
Sequence{1.2, 2.3, 5.2, 0.9}->any(self < 1)	0.9
Sequence{1.2, 2.3, 5.2, 0.9}->any(self < 2)	1.2

asBag () : Bag(T)

Returns a Bag containing all elements of *self*.

Expression	Result
Sequence{'3', 1, 2.0, '3'}->asBag()	Bag{2.0, '3', 1, '3'}
Bag{1, 2.0, '3'}->asBag()	Bag{2.0, 1, '3'}
OrderedSet{1, 2.0, '3'}->asBag()	Bag{2.0, 1, '3'}
OrderedSet{1, 1, 2.0, '3'}->asBag()	Bag{'3', 1, 2.0}
Set{1, 2.0, '3'}->asBag()	Bag{2.0, 1, '3'}
Set{1, 1, 2.0, '3'}->asBag()	Bag{2.0, '3', 1}

asOrderedSet () : OrderedSet(T)

Returns an OrderedSet containing all elements of *self*. Element ordering is preserved when possible.

Expression	Result
Sequence{1, 2.0, '3'}->asOrderedSet()	OrderedSet{1, '3', 2.0}
Sequence{1, 1, 2.0, '3'}->asOrderedSet()	OrderedSet{'3', 1, 2.0}
Bag{1, 2.0, '3'}->asOrderedSet()	OrderedSet{2.0, 1, '3'}
Bag{1, 1, 2.0, '3'}->asOrderedSet()	OrderedSet{1, '3', 2.0}
OrderedSet{1, 2.0, '3'}->asOrderedSet()	OrderedSet{1, 2.0, '3'}
Set{1, 2.0, '3'}->asOrderedSet()	OrderedSet{'3', 1, 2.0}

asSequence () : Boolean

Returns a Sequence containing all elements of *self*. Element ordering is preserved when possible.

Expression	Result
Sequence{1, 2.0, '3'}->asSequence()	Sequence{1, 2.0, '3'}
Bag{1, 2.0, '3'}->asSequence()	Sequence{2.0, 1, '3'}
OrderedSet{1, 2.0, '3'}->asSequence()	Sequence{1, 2.0, '3'}
Set{1, 2.0, '3'}->asSequence()	Sequence{'3', 1, 2.0}

asSet () : Set(T)

Returns a Set containing all elements of *self*.

Expression	Result
Sequence{1, 2.0, '3'}->asSet()	Set{1, '3', 2.0}
Sequence{1, 1, 2.0, '3'}->asSet()	Set{'3', 1, 2.0}
Bag{1, 2.0, '3'}->asSet()	Set{2.0, 1, '3'}
Bag{1, 1, 2.0, '3'}->asSet()	Set{1, '3', 2.0}
OrderedSet{1, 2.0, '3'}->asSet()	Set{1, '3', 2.0}
OrderedSet{1, 1, 2.0, '3'}->asSet()	Set{'3', 1, 2.0}
Set{1, 2.0, '3'}->asSet()	Set{2.0, 1, '3'}
Set{1, 1, 2.0, '3'}->asSet()	Set{'3', 1, 2.0}

collect (expr : OclExpression) : Collection(T2)

Returns a collection containing the result of applying *expr* on all elements contained in *self*.

Expression	Result
Sequence{'first', 'second'}->collect(toUpper())	Sequence{'FIRST', 'SECOND'}

collectNested (expr : OclExpression) : Collection(T2)

Returns a collection containing all the elements contained in *self* on which we applied the OclExpression *expr*. The results won't be flattened. The type of the resulting collection depends on the type of *self*.

For the purpose of these examples we'll assume here that we have a Class *Person* with a reference *children*. Our model contains two persons such as *person1.children = {James, Jane}* and *person2.children = {John}*.

Expression	Result
self.persons->collectNested(children.firstname)	Sequence{Sequence{James, Jane}, Sequence{John}}

count (object : T) : Integer

Returns how many times *object* is in the collection *self*.

Expression	Result
Sequence{2.3, 5.2}->count(5.2)	1
Set{3, 'test', 4.0, 4, 4.0, 'test'}->count(null)	0
Set{3, null, 4.0, null, 'test'}->count(null)	1
Bag{3, null, 4.0, null, 'test'}->count(null)	2

excludes (object : T) : Boolean

Returns **true** if *object* is not contained in *self*, **false** otherwise.

Expression	Result
Sequence{2.3}->excludes(2.1)	true
Sequence{2.0}->excludes(2)	false

excludesAll (c2 : Collection(T)) : Boolean

Returns **true** if no element of *c2* is contained in *self*, **false** otherwise.

Expression	Result
Sequence{2.3, 5.2, 'a', 3, null}->excludesAll(Set{4, null})	false
Sequence{2.3, 5.2, 'a', 3}->excludesAll(Set{4, null})	true

excluding (object : T) : Collection(T)

Returns a collection containing all elements of *self* minus all occurences of *object*. **Note** : at the time of writing, the OCL standard library sports a bug which changes *OrderedSets* in *Sets* when excluding elements.

Expression	Result
Sequence{'b', 'a', 'b', 'c'}->excluding('b')	Sequence{'a', 'c'}
Bag{'b', 'a', 'b', 'c'}->excluding('b')	Bag{'c', 'a'}
OrderedSet{'b', 'a', 'b', 'c'}->excluding('b')	Set{'c', 'a'}
Set{'b', 'a', 'b', 'c'}->excluding('b')	Set{'c', 'a'}

exists (expr : OclExpression) : Boolean

Returns **true** if at least one element in *self* validates the condition *expr*, **false** otherwise. The evaluation stops as soon as one element validating *expr* is found.

Expression	Result
Sequence{2.3, 5.2}->exists(self > 3)	true

flatten () : Collection(T2)

Returns a collection containing all elements of *self* recursively flattened. **Note** : at the time of writing, the OCL standard library sports a bug which changes *OrderedSets* in *Sets* when flattening.

Expression	Result
Sequence{Set{1, 2, 3}, Sequence{2.0, 3.0}, Bag{'test'}}->flatten()	Sequence{1, 2, 3, 2.0, 3.0, 'test'}
Bag{Set{Bag{'test', 2, 3.0}}, Sequence{OrderedSet{2.0, 3, 1}}}->flatten()	Bag{1, 2, 3, 2.0, 3.0, 'test'}
OrderedSet{Set{Bag{'test', 2, 3.0}}, Sequence{Set{2.0, 3, 1}}}->flatten()	Set{3.0, 2, 1, 3, 'test', 2.0}
Set{Set{Bag{'test', 2, 3.0}}, Sequence{OrderedSet{2.0, 3, 1}}}->flatten()	Set{3.0, 2, 1, 3, 'test', 2.0}

forAll (expr : OclExpression) : Boolean

Returns **true** if the all the elements contained in *self* validate the condition *expr*, **false** otherwise.

Expression	Result
Sequence{2.3, 5.2}->forAll(self > 3)	false
Sequence{2.3, 5.2}->forAll(self > 1.2)	true

includes (object : T) : Boolean

Returns **true** if *object* is contained in *self*, **false** otherwise.

Expression	Result
Sequence{2.3}->includes(2.1)	false
Sequence{2.0}->includes(2)	true

includesAll (c2 : Collection(T)) : Boolean

Returns **true** if all element of *c2* are contained in *self*, **false** otherwise.

Expression	Result
Sequence{2.3, 5.2, 'a', 3, null}->includesAll(Set{3, null})	true
Sequence{2.3, 5.2, 'a', 3}->includesAll(Set{3, null})	false

including (object : T) : Collection(T)

Returns a collection containing all elements of *self* followed by *object*. **Note** : at the time of writing, the OCL standard library sports a bug which changes *OrderedSets* in *Sets* when including elements.

Expression	Result
Sequence{'a', 'b'}->including('c')	Sequence{'a', 'b', 'c'}
Bag{'a', 'b'}->including('c')	Bag{'a', 'c', 'b'}
OrderedSet{'a', 'b'}->including('c')	Set{'a', 'c', 'b'}
Set{'a', 'b'}->including('c')	Set{'a', 'c', 'b'}

isEmpty () : Boolean

Returns **true** if *self* is empty, **false** otherwise.

Expression	Result
Sequence{2, 'a'}->isEmpty()	false
Sequence{null}->isEmpty()	false
Sequence{}->isEmpty()	true

isUnique (expr : OclExpression) : Boolean

Returns **true** if all elements contained in *self* evaluate to a distinct value for *expr*.

Expression	Result
Sequence{2.3, 5.2}->isUnique(self > 3)	true
Sequence{2.3, 5.2}->isUnique(self > 1)	false

notEmpty () : Boolean

Returns **true** if *self* contains at least one element, **false** otherwise.

Expression	Result
Sequence{2, 'a'}->notEmpty()	true
Sequence{null}->notEmpty()	true
Sequence{}->notEmpty()	false

one (expr : OclExpression) : Boolean

Returns **true** if there is only one element contained in *self* that validates the condition *expr*, **false** otherwise.

Expression	Result
Sequence{1.2, 2.3, 5.2, 0.9}->one(self < 1)	true
Sequence{1.2, 2.3, 5.2, 0.9}->one(self < 2)	false

product (c2 : Collection(T2)) : Set(Tuple(first : T, second : T2))

Returns a Set of Tuples which represents the cartesian product of *self* with *c2*.

Expression	Result
Sequence{3, 4}->product(Bag{3.0, 4.0})	Set{Tuple{3, 3.0}, Tuple{3, 4.0}, Tuple{4, 3.0}, Tuple{4, 4.0}}
Set{3, 4}->product(OrderedSet{3.0, 4.0})	Set{Tuple{3, 3.0}, Tuple{3, 4.0}, Tuple{4, 3.0}, Tuple{4, 4.0}}

reject (expr : OclExpression) : Collection(T)

Returns a collection with all elements of *self* except for those who validate the OclExpression *expr*.

Expression	Result
i > 1)	Sequence{1}
i > 1)	Bag{1}
i > 1)	OrderedSet{1}
i > 1)	Set{1}

select (expr : OclExpression) : Collection(T)

Returns a collection with all elements of *self* that validate the OclExpression *expr*.

Expression	Result
i > 1)	Sequence{2, 3}
i > 1)	Bag{3, 2}
i > 1)	OrderedSet{2, 3}
i > 1)	Set{3, 2}

size () : Integer

Returns the number of elements contained in *self*.

Expression	Result
Sequence{2.3, 5}->size()	2
Sequence{}->size()	0

sortedBy (expr : OclExpression) : Sequence(T)

Returns a sorted collection containing all elements from *self* sorted in accordance with the OclExpression *expr*. This can be used on all kind of collections yet will always yield a Sequence-typed result except for OrderedSet which returns an OrderedSet.

For the purpose of these examples we'll assume here that we have a Class *Employee* with an attribute *age*. Our model contains two employees such as *employee1.age = 24* and *employee2.age = 27*.

Expression	Result
self.employees->sortedBy(age)	Sequence{employee1, employee2}

sum () : Real

Returns the sum of all elements contained in *self* if they support the '+' operation.

Expression	Result
Sequence{2.3, 5.2} in c->sum()	7.5
Sequence{2, 4} in c->sum()	6
Sequence{2, '4'} in c->sum()	***invalid***

Ocl operations for type *Sequence*

= (seq : Sequence(T)) : Boolean

Returns **true** if *self* contains the very same objects as *seq* in the very same order as they are in *seq*.

Expression	Result
Sequence{4, 5, 'test'} = Sequence{4, 5, 'test'}	true
Sequence{4, 5, 'test'} = Sequence{4, 'test', 5}	false
Sequence{4, 5, 'test', 5} = Sequence{4, 5, 'test'}	false

<> (seq : Sequence(T)) : Boolean

Returns **true** if *self* does not contain the same objects as *seq*, or if these objects are not in the same order as they are in *seq*.

Expression	Result
Sequence{4, 5, 'test'} = Sequence{4, 5, 'test'}	false
Sequence{4, 5, 'test'} = Sequence{4, 'test', 5}	true
Sequence{4, 5, 'test', 5} = Sequence{4, 5, 'test'}	true

append (object : T) : Sequence(T)

Returns a Sequence containing all elements of *self* followed by *object*.

Expression	Result
Sequence{'a', 'b'}->append('c')	Sequence{'a', 'b', 'c'}

at (index : Integer) : T

Returns the element of *self* at the *index* position.

Expression	Result
Sequence{'a', 'b'}->at(1)	a

first () : T

Returns the first element of *self*.

Expression	Result
Sequence{1, 2.0, '3'}->first()	1

indexOf (object : T) : Integer

Returns the position of *object* in sequence *self*.

Expression	Result
Sequence{'a', 'b'}->indexOf('a')	1

insertAt (index : Integer, object : T) : Sequence(T)

Returns a Sequence containing *self* with *object* inserted at the *index* position.

Expression	Result
Sequence{'a', 'b'}->insertAt(0, 'c')	***invalid***
Sequence{'a', 'b'}->insertAt(1, 'c')	Sequence{'c', 'a', 'b'}
Sequence{'a', 'b'}->insertAt(3, 'c')	Sequence{'a', 'b', 'c'}
Sequence{'a', 'b'}->insertAt(4, 'c')	***invalid***

last () : T

Returns the last element of *self*.

Expression	Result
Sequence{1, 2.0, '3'}->last()	'3'

prepend (object : T) : Sequence(T)

Returns a Sequence containing *object* followed by all elements of *self* .

Expression	Result
Sequence{'a', 'b'}->prepend('c')	Sequence{'c', 'a', 'b'}

subSequence (startIndex : Integer, endIndex : Integer) : Sequence(T)

Returns a Sequence containing all elements of *self* between the positions 'startIndex' and 'endIndex'.

Expression	Result
Sequence{'a', 'b', 'c', 'd'}->subSequence(2, 3)	Sequence{'b', 'c'}
Sequence{'a', 'b', 'c', 'd'}->subSequence(4, 4)	Sequence{'d'}

union (seq : Sequence(T)) : Sequence(T)

Returns a Sequence containing all elements of *self* followed by all elements of *seq*.

Expression	Result
Sequence{'a', 'b', 'a'}->union(Sequence{'b', 'c'})	Sequence{'a', 'b', 'a', 'b', 'c'}

Ocl operations for type *Bag*

= (bag : Bag(T)) : Boolean

Returns **true** if *self* contains the same objects as *bag* in the same quantities.

Expression	Result
Bag{4, 5, 'test', 4} = Bag{4, 'test', 5, 4}	true
Bag{4, 5, 'test'} = Bag{4, 'test', 5}	true
Bag{4, 5, 'test', 5} = Bag{4, 5, 'test'}	false

<> (bag : Bag(T)) : Boolean

Returns **true** if *self* does not contain the same objects as *bag* in the same quantities.

Expression	Result
Bag{4, 5, 'test'} = Bag{4, 5, 'test'}	false
Bag{4, 5, 'test'} = Bag{4, 'test', 5}	false
Bag{4, 5, 'test', 5} = Bag{4, 5, 'test'}	true

intersection (bag : Bag(T)) : Bag(T)

Returns a Bag containing all elements of *self* that are also contained by *bag*.

Expression	Result
Bag{'a', 'b', 'a'}->intersection(Bag{'a', 'b'})	Bag{'a', 'b'}
Bag{'a', 'b', 'a', 'b'}->intersection(Bag{'a', 'b', 'b'})	Bag{'b', 'a', 'b'}

intersection (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* that are also contained by *set*.

Expression	Result
Bag{'a', 'b', 'a'}->intersection(Set{'a', 'b', 'c'})	Set{'a', 'b'}

union (bag : Bag(T)) : Bag(T)

Returns a Bag containing all elements of *self* and all elements of *bag*.

Expression	Result
Bag{'a', 'b', 'a'}->union(Bag{'b', 'c'})	Bag{'b', 'a', 'b', 'a', 'c'}

union (set : Set(T)) : Bag(T)

Returns a Bag containing all elements of *self* and all elements of *set*.

Expression	Result
Bag{'a', 'b', 'a'}->union(Set{'b', 'c'})	Bag{'b', 'c', 'a', 'b', 'a'}

Ocl operations for type *OrderedSet*

= (set : Set(T)) : Boolean

Returns **true** if *self* contains the same objects as *set*.

Expression	Result
OrderedSet{3, 5, 4} = Set{3, 5, 4}	true
OrderedSet{3, 5, 4} = Set{4, 3, 5, 4, 4}	true
OrderedSet{3, 5, 4} = Set{2, 5 ,4, 4}	false

= (orderedset : OrderedSet(T)) : Boolean

Returns **true** if *self* contains the same objects as *orderedset* regardless of element ordering.

Expression	Result
OrderedSet{3, 5, 4} = OrderedSet{3, 5, 4}	true
OrderedSet{4, 5, 'test', 5} = OrderedSet{4, 5, 'test'}	true
OrderedSet{4, 5, 'test'} = OrderedSet{4, 'test', 5}	true
OrderedSet{4, 5, 'test'} = OrderedSet{4, 'test'}	false

<> (set : Set(T)) : Boolean

Returns **true** if *self* does not contain the same objects as *set*.

Expression	Result
OrderedSet{4, 5, 'test', 4} <> Set{4, 5, 'test'}	false
OrderedSet{4, 5, 'test', 4} <> Set{4, 'test', 5, 4}	false
OrderedSet{4, 5, 'test', 4} <> Set{4, 5, 'test', 2}	true

<> (orderedset : OrderedSet(T)) : Boolean

Returns **true** if *self* does not contain the same objects as *orderedset*.

Expression	Result
OrderedSet{4, 5, 'test', 4} <> OrderedSet{4, 5, 'test')	false
OrderedSet{4, 5, 'test', 4} <> OrderedSet{4, 'test', 5, 4}	false
OrderedSet{4, 5, 'test', 4} <> OrderedSet{4, 5, 'test', 2}	true

`-` (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* minus all elements of *set*.

Expression	Result
OrderedSet{'a', 'b', 'c'} - Set{'c', 'a'}	Set{'b'}

append (object : T) : OrderedSet(T)

Returns an OrderedSet containing all elements of *self* followed by *object*.

Expression	Result
OrderedSet{'a', 'b'}->append('c')	OrderedSet{'a', 'b', 'c'}

at (index : Integer) : T

Returns the element of *self* located at position *index* in the collection.

Expression	Result
OrderedSet{'a', 'b'}->at(1)	'a'

first () : T

Returns the first element of *self*.

Expression	Result
OrderedSet{1, 2.0, '3'}->first()	1

indexOf (object : T) : Integer

Returns the position of *object* in *self*.

Expression	Result
OrderedSet{'a', 'b'}->indexOf('a')	1

insertAt (index : Integer, object : T) : OrderedSet(T)

Returns an OrderedSet containing *self* with *object* inserted at the *index* position.

Expression	Result
OrderedSet{'a', 'b'}->insertAt(1, 'c')	OrderedSet{'c', 'a', 'b'}
OrderedSet{'a', 'b'}->insertAt(3, 'c')	OrderedSet{'a', 'b', 'c'}

intersection (bag : Bag(T)) : Set(T)

Returns a Set containing all elements of *self* that are also contained by *bag*.

Expression	Result
OrderedSet{'a', 'b', 'a'}->intersection(Bag{'a', 'b'})	Set{'a', 'b'}

intersection (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* that are also contained by *set*.

Expression	Result
OrderedSet{'a', 'b', 'a'}->intersection(Set{'a', 'b'})	Set{'a', 'b'}

last () : T

Returns the last element of *self*.

Expression	Result
OrderedSet{1, 2.0, '3'}->last()	'3'

prepend (object : T) : OrderedSet(T)

Returns an OrderedSet containing *object* followed by all elements of *self*.

Expression	Result
OrderedSet{'a', 'b'}->prepend('c')	OrderedSet{'c', 'a', 'b'}

subOrderedSet (startIndex : Integer, endIndex : Integer) : OrderedSet(T)

Returns an OrderedSet containing all elements of *self* between the positions *startIndex* and *endIndex*.

Expression	Result
OrderedSet{'a', 'b', 'c', 'd'}->subOrderedSet(2, 3)	OrderedSet{'b', 'c'}
OrderedSet{'a', 'b', 'c', 'd'}->subOrderedSet(4, 4)	OrderedSet{'d'}

symmetricDifference (set : Set(T)) : Set(T)

Returns a Set containing all of the elements of *self* and *set* that are not present in both.

Expression	Result
OrderedSet{'b', 'a', 'b', 'c'}->symmetricDifference(Set{'a', 'c', 'd'})	Set{'d', 'b'}

union (bag : Bag(T)) : Bag(T)

Returns a Bag containing all elements of *self* followed by all elements of *bag*.

Expression	Result
OrderedSet{'a', 'b', 'a'}->union(Bag{'b', 'c'})	Bag{'a', 'c', 'b', 'b'}

union (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* followed by all elements of *set*.

Expression	Result
OrderedSet{'a', 'b', 'a'}->union(Set{'b', 'c'})	Set{'a', 'c', 'b'}

Ocl operations for type *Set*

= (set : Set(T)) : Boolean

Returns **true** if *self* contains the same objects as *set*.

Expression	Result
Set{3, 5, 4} = Set{3, 5, 4}	true
Set{3, 5, 4} = Set{3, 4, 4, 5}	true
Set{3, 5, 4} = Set{2, 3, 5, 4}	false

<> (set : Set(T)) : Boolean

Returns **true** if *self* does not contain the same objects as *set*.

Expression	Result
Set{4, 5, 'test', 4} <> Set{4, 5, 'test'}	false
Set{4, 5, 'test', 4} <> Set{5, 4, 'test', 4}	false
Set{4, 5, 'test', 4} <> Set{4, 'test', 5, 2}	true

`-` (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* minus all elements of *set*.

Expression	Result
Set{'a', 'b', 'c'} - Set{'c', 'a'}	Set{'b'}

intersection (bag : Bag(T)) : Set(T)

Returns a Bag containing all elements of *self* that are also contained in *bag*.

Expression	Result
Set{'a', 'b', 'a'}->intersection(Bag{'a', 'b', 'c'})	Set{'a', 'b'}

intersection (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* that are also contained in *set*.

Expression	Result
Set{'a', 'b', 'a'}->intersection(Set{'a', 'b', 'c'})	Set{'b', 'a'}

symmetricDifference (set : Set(T)) : Set(T)

Returns a Set containing all of the elements of *self* and *set* that are not present in both.

Expression	Result
Set{'b', 'a', 'b', 'c'}->symmetricDifference(Set{'a', 'c', 'd'})	Set{'b', 'd'}

union (bag : Bag(T)) : Bag(T)

Returns a Bag containing all elements of *self* and all elements of *bag*.

Expression	Result
Set{'a', 'b', 'a'}->union(Bag{'b', 'c'})	Bag{'a', 'c', 'b', 'b'}

union (set : Set(T)) : Set(T)

Returns a Set containing all elements of *self* and all elements of *set*.

Expression	Result
Set{'a', 'b', 'a'}->union(Set{'b', 'c'})	Set{'a', 'c', 'b'}

Ocl operations for type *Boolean*

And

And	True	False	Invalid
True	true	false	true
False	false	false	false
Invalid	invalid	false	invalid

Implies

Implies	True	False	Invalid
True	true	false	invalid
False	true	true	true
Invalid	true	invalid	invalid

Or

Or	True	False	Invalid
True	true	true	true
False	true	false	invalid
Invalid	true	invalid	invalid

Not

Not	Result
True	False
False	True
Invalid	invalid

Xor

Xor	True	False	Invalid
True	false	true	invalid
False	true	false	invalid
Invalid	invalid	invalid	invalid