Object Constraint Language (OCL) Cheat Sheet

OCL SYNTAX

cs	e op e		ns:: ... ns::id *(7.5.7)*
id	e . id	*(7.4.10)*	**if** pd **then** e **else** e **endif**
self	e . pt (e, ... , e)	*(7.4.10)*	**let** id = e : T, id2 = e:T, ... **in** e2 *(7.4.3*
	c **->** pt (e, ..., e)	*(7.4.10)*	

OCL LIBRARY

Type	Examples	Operations
Integer *(11.5.2)*	1, -5, 34	i+i2, i-i2, i*i2, i.div(i2), /, i.mod(i), i.abs(), i.max(i2), i.min(i2), <, >, <=, >=, i.toString()
Real *(11.5.1)*	1.5, 1.34, ...	r+r2, r-r2, r*r2, r/r2, r.floor, r.round(), r.max(r2), r.min(r2), <, >, <=, >=, r.toString()
Boolean *(11.5.4)*	true, false	not b, b and b2, b or b2, b xor b2, b implies b2, b.toString()
String *(11.5.3)*	'', 'a chair'	+, s.size(), s.concat(s2), s.substring(i1,i2), s.toInteger(), s.toReal(), s.toUpperCase(), s.toLowerCase(), s.indexOf(s2), s.equalsIgnoreCase(s2), s.at(i), s.characters(), s.toBoolean(), <, >, <=, >=
Enumeration *(7.4.2)*	Day::monday, Day::tuesday, …	=, <>
TupleType(**x : T1,** **y : T2,** **z : T3)** *(7.5.15)*	Tuple { y = 12 x = true, z:Real= 3.5 }	t.x t.y t.z
Collection(T) *(11.7.1)*		=, <>, c->size(), c->includes(o), c->excludes(o), c->count(o), c->includesAll(c2) c->excludesAll(c2), c->isEmpty(), c->notEmpty(), c->max(), c->min(), c->sum(), c->product(c2), c->selectByKind(ty), c->selectByType(ty), c->asSet(), c->asOrderedSet(), c->asSequence(), c->asBag(), c->flatten(), *(11.9.1)* c->any(it|pd), c->closure(it|e), c->collect(it|e), c->collectNested(it|e), c->exists(it1,it2...|pd), c->forall(it1,it2...|pd), c->isUnique(it|e), c->one(it|pd), c->reject(it|pd), c->select(it|pd), c->sortedBy(it|e), c->iterate(e|)
Set(T) *(11.7.2)*	Set {1,5,10,3}, Set{}	st->union(st2), st->union(bg), st->intersection(st2), st->intersection(bg) st - st2, st->including(e), st->excluding(e), st->symmetricDifference(st2)
Bag(T) *(11.7.4)*	Bag {1,5,5} Bag {}	bg->union(bg2), bg->union(st), bg->intersection(bg2), bg->intersection(st) bg->including(e), bg->excluding(e)
OrderedSet(T) *(11.7.13)*	OrderedSet{10,4,3} OrderedSet{}	os->append(e), os->prepend(e), os->insertAt(e), os->subOrderedSet(i1,i2), os->at(i), os->indexOf(e), os->first(), os->last(), os->reverse()
Sequence(T) *(11.7.4)*	Sequence{5,3,5} Sequence{}	sq->union(sq2), sq->append(e), sq->prepend(e), sq->insertAt(i,o) sq->subSequence(i1,i2), sq->at(i), sq->indexOf(o), sq->first(), sq->last(), sq->including(e), sq->excluding(e), sq->reverse()
Class		cl.allInstances()
Global functions		e.oclIsTypeOf(ty), e.oclIsKindOf(ty), e.oclAsType(ty) e.oclIsInState(state), e.oclIsNew()

i : Integer	c : Collection(T)	os : OrderedSet(T)	cs: constant	ty : type
r : Real	st: Set(T)	t : Tuple(…)	pd : predicat	it : iterator
b : Boolean	bg : Bag(T)	id: identificateur	e : expression	cl : classifier
s : String	sq : Sequence(T)	pt: property	ns: namespace	

```
(age<40 implies salary>1000) and (age>=40 implies salary>2000)
salary > (if age<40 then 1000 else 2000 endif)
name = name.substring(1,1).toUpper().concat(name.substring(2,name.size()).toLower())
let s:integer = 2000 in s*s+s
Set{3,5,2, 45, 5 }->union(Set{2,8,2})->size()
Sequence{1,2,45,9,3,9}->count(9) + (if Sequence{1,2,45,2,3,9}->includes(45) then 10 else 2)
Sequence{1..Set{7,8}->max()}->includes(6)
Bag{1,9,9,1} -> count(9)
c->asSet()->size() = c->size()
Tuple{name='bob',age=18}.age
Set{2,3}->product(Set{'a','b'})->includes(Tuple{first=2,second='b'})
self.children.children.firstnames = Bag{'pierre','paul','marie','paul'}
self.children->select( age>10 and sexe = Sex::Male)
self.children->reject( p | p.children->isEmpty())->notEmpty()
self.members->any(title='president')
self.children->forall( e | e.age < self.age - 7)
self.children->forall( e : Person | e.age < self.age - 7)
self.children->forall( e1,e2 : Person |  e1 <> e2 implies e1.name <> e2.name)
self.children->isUnique( name )
self.parents.parents.children.children->excluding(parents.children)->asSet()
self.children.children.firstname = Bag{'pierre','marie','pierre'}
self.children->collect(c|c.children.firstname)=Bag{Bag{'pierre'},Bag{'marie','pierre'}
self.children->collectNested(c|c.children.firstname) = Bag{Bag{'pierre'},Bag{'marie','pierre'}
self.spouse->notEmpty() implies spouse.sex = Sex::Female
Sequence{2,5,3}->collect(i|i*i+1) = Sequence{5,26,10}
enfants->sortedBy( age )
enfants->sortedBy( enfants->size() )->last()
let ages = enfants.age->sortedBy(a | a) in ages.last() - ages.first()
Set{1,2,3}->iterate(e;acc:Integer=0|acc+e)
```

USE SPECIFICATION LANGUAGE (.use)

```
model JungleExample
-- ========================== Enumerations ====================================
enum Season {winter, autumn, spring, summer}   -- example of value: Season::winter
-- ========================== Classes =========================================
class Fruit end              -- Classes used below
class ForestThing end
class Animal end
-- Class, Inheritance, Attributes, Operations, Local constraints
class Banana < Fruit, ForestThing   -- Banana inherits from Fruit and ForestThing
attributes
    length : Integer /* Integer, Real, Boolean, String */
    growthTime : Season
    -- Tuple, Bag, Set, OrderedSet, Sequence
    goodies : OrderedSet(Bag(Sequence(Set(Tuple(x:Integer,y:Real,z:String)))))
    -- Attribute initialisation
    remainingDays : Integer
        init: 0
    -- Derived attribute
    size : Real
        derived: self.length * self.remainingDays
    -- RESTRICTION/std: No invariants directly declared on attributes
    -- RESTRICTION/std: No cardinality supported for attributes (e.g. String[0..1])
operations
    wakeUp(n : Integer):String       -- operation specified
        pre notTooMuch: n > 10 and n < self.length   -- precondition
        post resultOk: result > 'anaconda'           -- postcondition
    helloJungle() : String           -- operation with soil actions
        begin
            declare x : Banana ;
            WriteLine('hello') ;
            x := new Banana ;
            self.length := self.length + self.remainingDays*20+3 ;
            result := 'jungle' ;
            destroy x ;
        end
        pre freshEnough: self.remainingDays > 10
    smash() : String                 -- operation/query defined in OCL
        = 'li'+'on' -- derived/query operation defined as anOCL expression
constraints
    -- invariants
    inv growthSeasons: Set{Season::summer,Season::winter}->includes(self.growthTime)
end -- end of class Banana
-- ========================== Associations ====================================
-- Associations, Roles, Cardinality
association Eats between   -- 'association' or 'composition' or 'aggregation'
    Animal[*] role eater   -- could be followed by 'ordered'
    Banana[1] role food    -- cardinality can be [1..8,10,15..*]
    -- ...                 -- more roles here for n-ary associations
end
-- Association classes
associationclass Dislike between
    Animal [0..1] role animal
    Banana[1..*] role bananas
attributes                 -- operations can be declared as well
    reason : String
end
-- Qualified associations
association Prefers between
    Animal [*] role animals qualifier (period:Season)
    Fruit[0..1] role candy
end
-- ========================== External Constraints ============================
constraints
context Banana                                  -- Constraints on Classes
    inv atLeastOne: Banana.allInstances()->size()>1
context self:Banana                             -- Constraints on Attributes
    inv largeEnough: self.length > 3
context Banana::wakeUp(n:Integer):String        -- Constraints on Operations
    -- Constraints on Operations
    pre justOk: self.length < 1000 and n > 12
    post notTiger: result <> 'tiger'
```

SOIL ACTION LANGUAGE (.soil)

```
open -q background.soil                          -- include a file
? Set{2,3}->including(7)                         -- OCL query
! b1 := new Banana ; chita := new Animal         -- object creation
! insert(chita,b1) into Eats                     -- link creation
! d := new Dislike between (chita,b1)            -- object-link creation(class association)
! b1.length := 20                                -- attribute assignment
? b1.smash()+' are nices'                        -- call of a query (defined in OCL)*
! destroy d                                      -- object/object-link destruction
! delete (chita,b1) from Eats                    -- link destruction
! Write('jungle'+(4+2).toString()) ; WriteLine('')   -- output
! r := ReadLine() ; i := ReadInteger() ;         -- input
! if not (b1.length=20) then WriteLine('error1') end  -- if then else
! for i in Sequence{1..4} do b := new Banana ; insert(chita,b) into Eats end
```

```
Commands :

?  (queries)
!  (actions)
open
check
info vars
info state
reset
help
quit

\ .(multiline cmd)
```