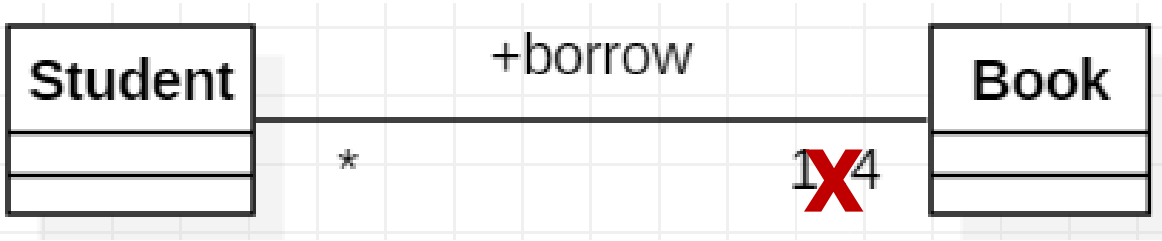
Object-Oriented Analysis

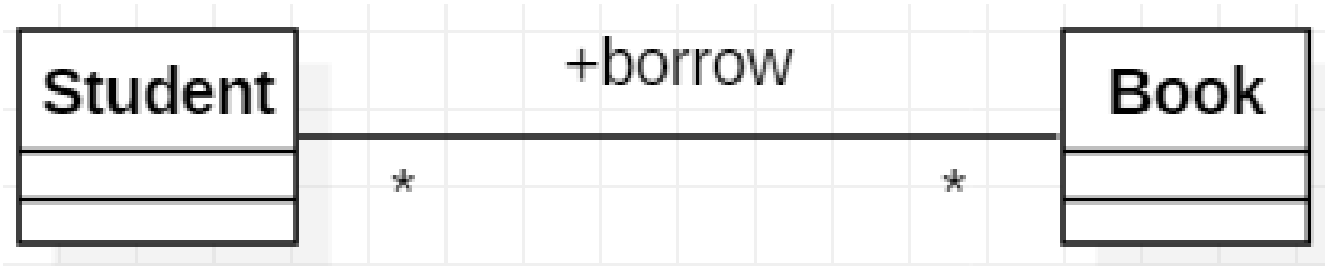
Lecture 4 : Object Constraint Language (OCL)

Nga Nguyen, EISTI

Example



Example



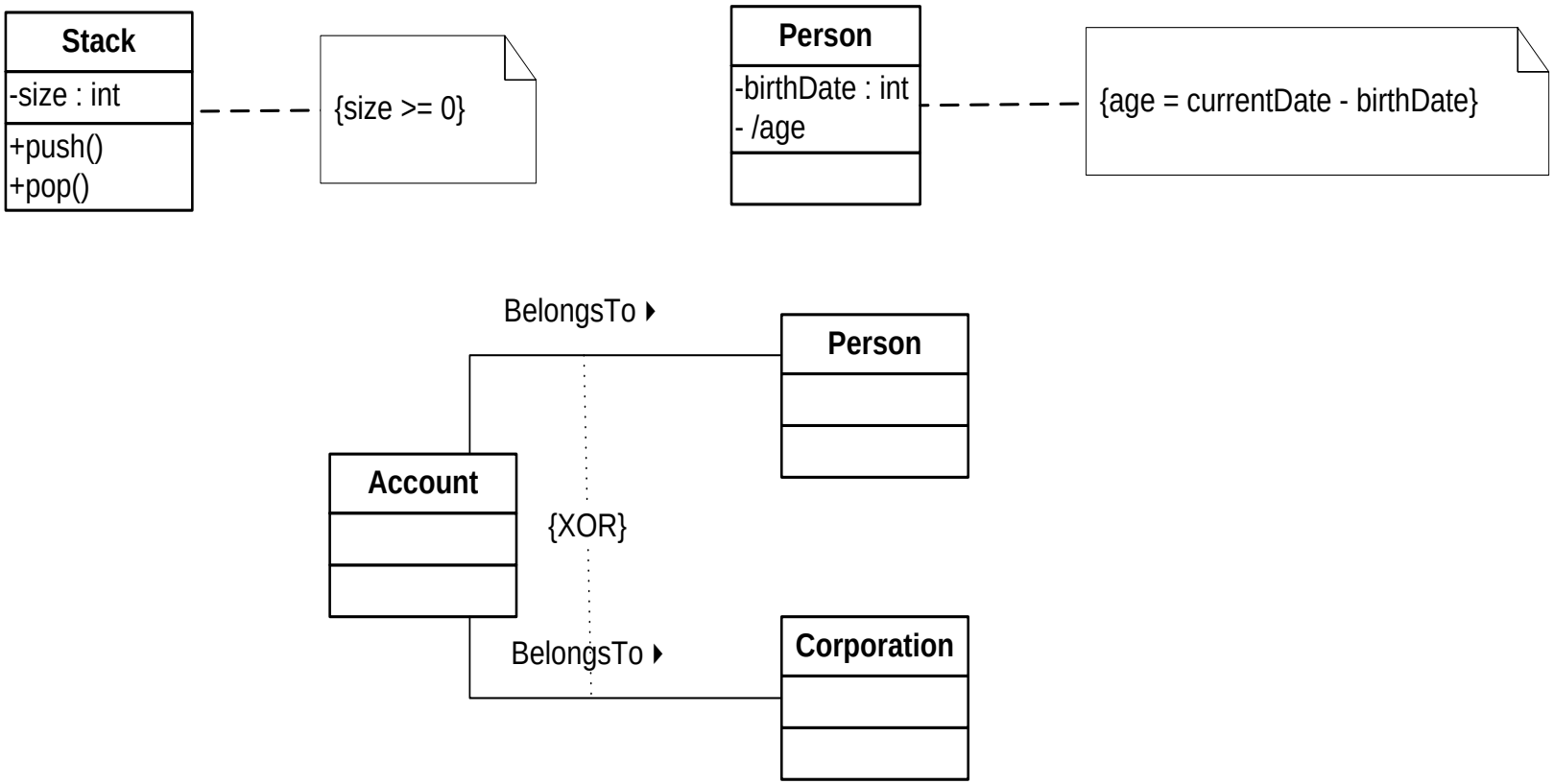
What is OCL?

- Describe constraints in UML models.
- Formal language (no ambiguity): read and understood by developers and clients
- Declarative language: *what* but not *how ?*
- Last version : OCL 2.4 (December 201)

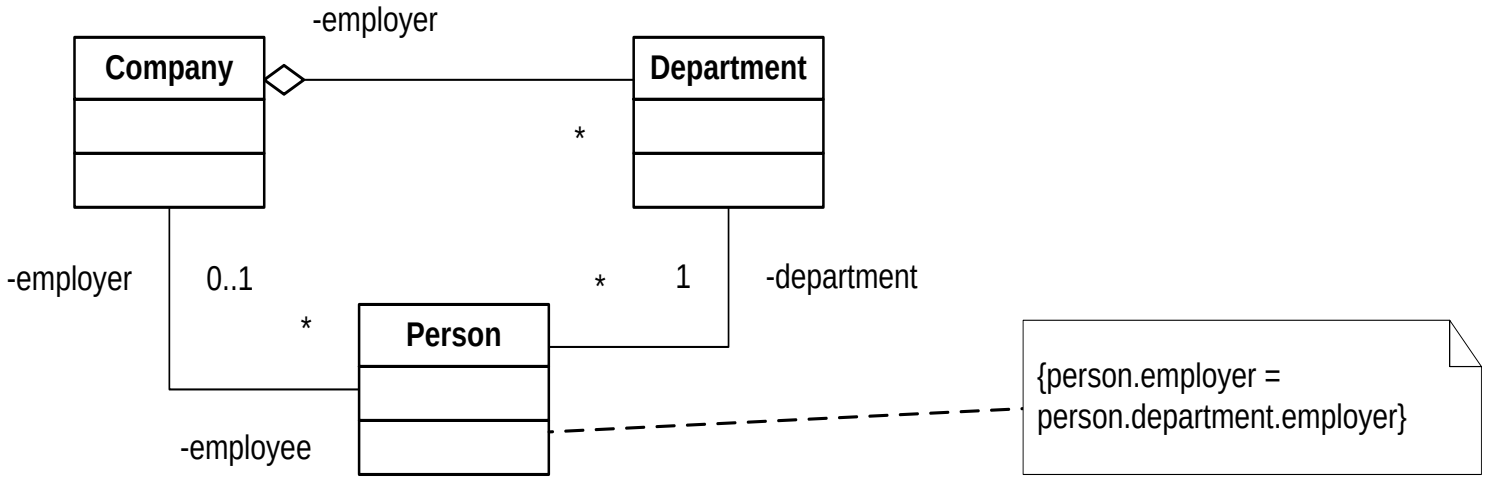
What is an OCL constraint?

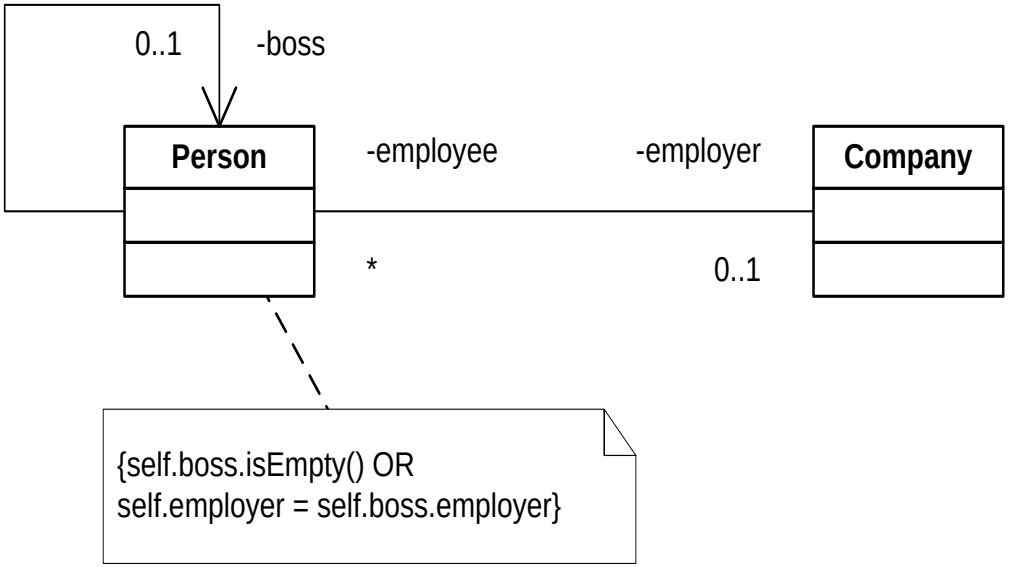
- "A constraint is a restriction on one or more values of (part of) an object-oriented model or system"
- A Boolean expression: *true* or *false*
- In a note/text/constraint editor in a UML diagram
  - Syntax : `{constraint}`
- Three types of constraint:
  - Invariant
  - Precondition
  - Postcondition

OCL constraints: examples



OCL constraints: examples





OCL constraint: Context

- An OCL expression is always associated to a model element (class, attribute, operation, relation, …): it is the context of the constraint
- There are two ways to specify the context of an OCL constraint :
  1) By writing the constraint between braces { } in a note. The element linked to this note will be the context.
  2) By using the keyword **`context`** in a separated document that goes with the diagram.

OCL : context

- Syntax : **`context`** `element`
- **Examples**
  - The context is the class *Account*:

    `context` `Account`
  - The context is the operation *getBalance()* of the class *Account*:

    `context` `Account::getBalance()`

Constraints : invariants

- An **invariant** is a constraint that should be true for an object during its complete lifetime

- Example :

```
context Account
inv : balance >= 0
```

Constraints : invariants

- An OCL expression OCL is always evaluated for a specific instance
  - Omitted naming:

```
context Person
inv : age >= 18
```

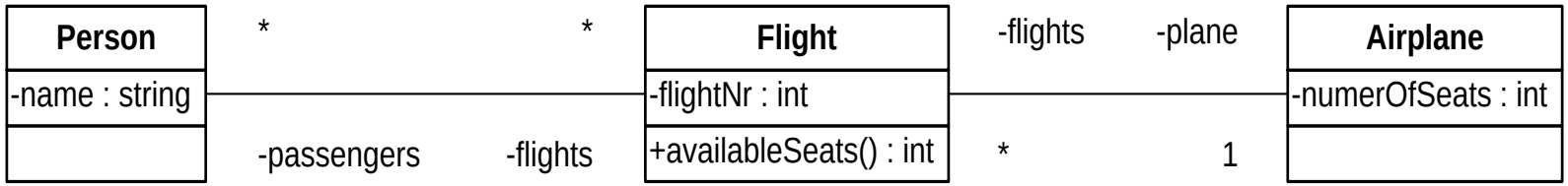
  - Naming by default: keyword *self*

```
context Person
inv : self.age >= 18
```

  - Explicit naming:

```
context p : Person
inv : p.age >= 18
```

Referring to properties of a class



- Notation "."
- Example: if Flight is the context, to refer to:
  - an attribute : `self.flightNr`
  - an operation : `self.availableSeats()`
  - another side of the association : `self.plane`
- The importance of roles!

Contraints : pre/post conditions

- We can specify pre/post conditions for the operations
  - preconditions must be true before calling the operation
  - post-conditions must be true after calling the operation

- In the post-conditions, we can use:
  - result : the return value
  - @pre : the value of an attribute before calling the operation

Constraints : example

```
context Account::debit(amount : Integer)
pre: amount > 0
post: balance = balance@pre – amount


context Account::getBalance(): Integer
post: result = balance
```

Naming the constraints

- Syntax :

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

- Examples :

```
context Account
inv positiveBalance: self.balance > 0

context Account::debit(amount : Integer)
pre positiveAmount : amount > 0
post amountDebited : self.balance =
self.balance@pre – amount
```

Comments

- Syntax :

```
-- comment
```

- Examples :

```
context Account
inv : self.balance > 0  -- positive balance

context Account ::debit(amount: Integer)
pre : amount > 0  -- positive amount
post: self.balance = self.balance@pre – amount
```

Result of an operation

- An OCL expression can be used to show the result of an "query" operation:

```
context TypeName::operation(param1:Type1,…): retType
body:--expression that returns un object of type
retType
```

- Example :

```
context Compte::getBalance() : Float
body : balance
```

Initial values

- An OCL expression can be used to show an initial value of an attribute

```
context TypeName::AttributeName: Type
init: -- expression representing the initial value
```

- Example :

```
context Person::married : Boolean
init: false
```

Derived values

- An OCL expression can be used to show the derived value of an attribute

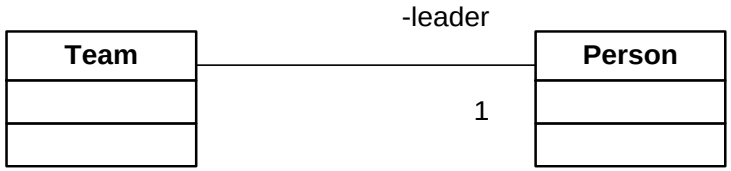
```
context TypeName::AttributeName: Type
derive:--expression representing the derived value
```

- Example :

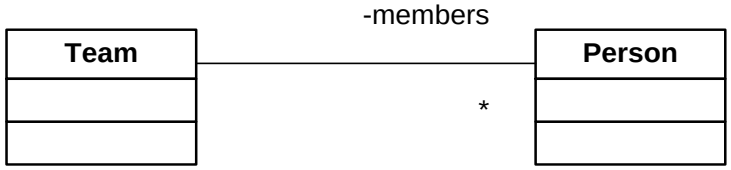
```
context Person::age : Integer
derive: Date::current() – birthday
```

Navigability and Collections

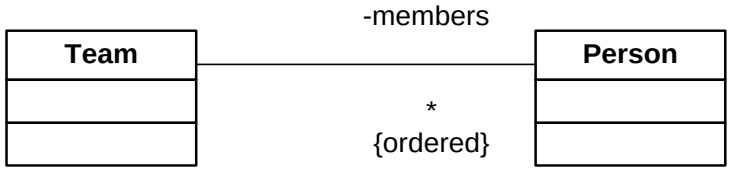
- In most cases, the navigation result is not an object but a collection of objects




```
context Team
self.leader : Person
```

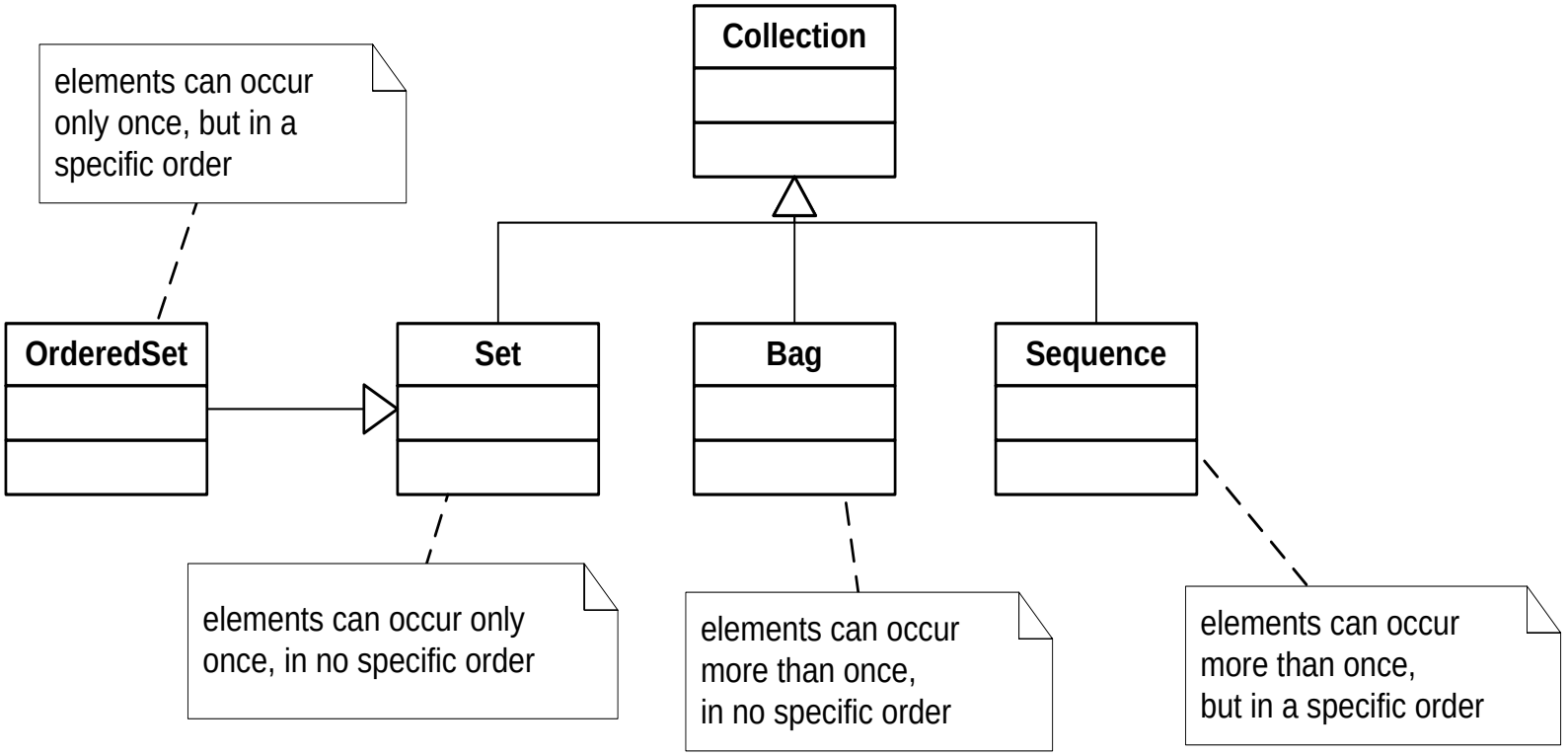



```
self.members : Set(Person)
```




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

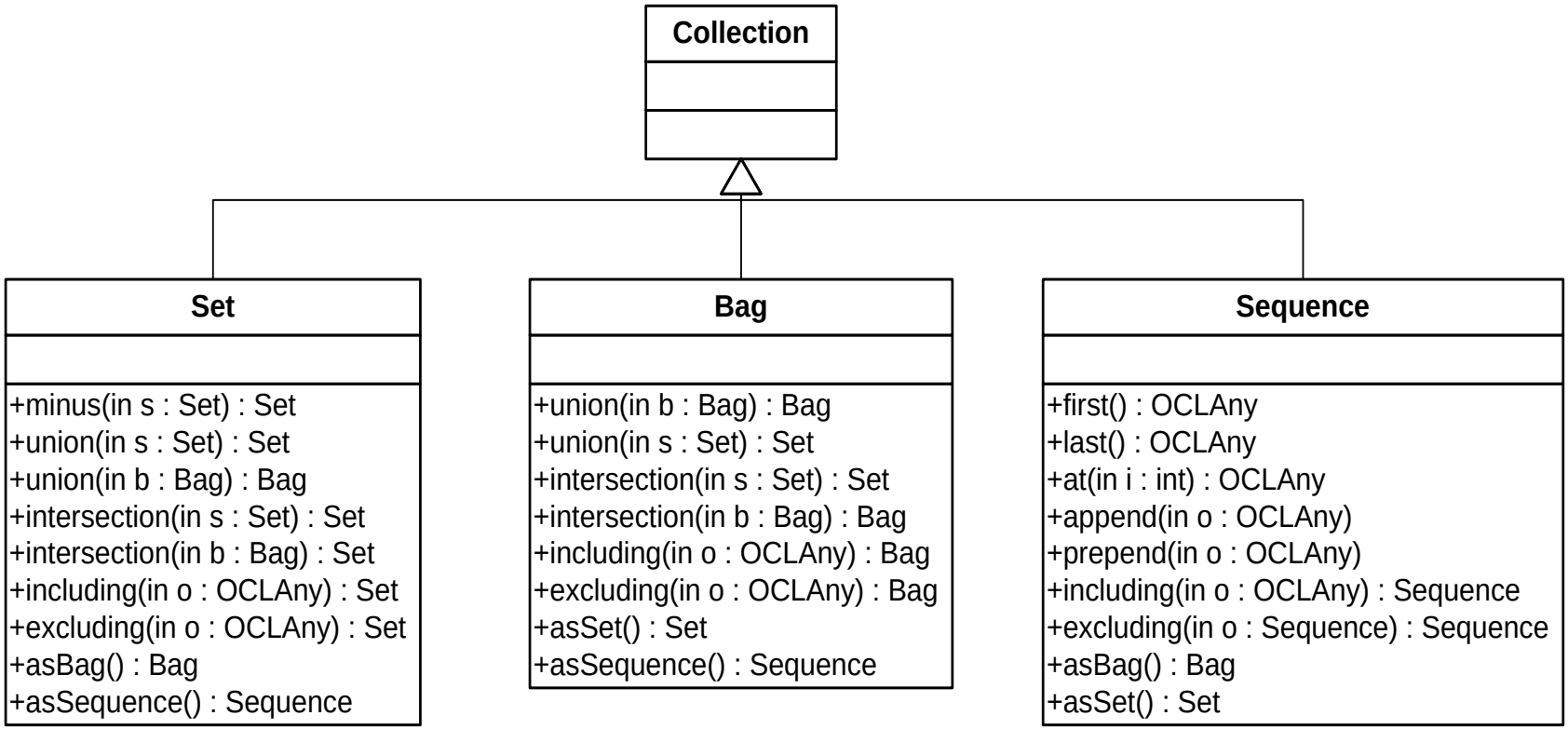
OCL collections



Collection operations

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([redacted])	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

Specialized operations



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}

Operations : examples (1)

- size() : Integer

```
context Company
inv : self.team->size() > 5
```

- isEmpty() : Boolean

```
context Person
inv : self.employer->isEmpty()
```

- notEmpty() : Boolean

```
context Company
inv : self.employees->notEmpty()
```

Operations : examples (2)

- includes(object : T) : Boolean

```
context Bank
-- the owner of the bank is one of its clients
inv : self.clients->includes(self.owner)
```

- excludes(object : T) : Boolean

```
context Bank
-- the owner of the bank cannot be a client
inv : self.clients->excludes(self.owner)
```

Operations : examples (3)

- count(object : T) : Boolean

```
context ChessBoard
inv : self.pieces->count(k : Knight) = 4
```

- sum() : T

```
context Student
inv : self.notes->sum()/self.exams->size()>10
```

Operations : examples (4)

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

Conditional constraints

- Syntax :
  - if expr1 then expr2 else expr3 endif
  - expr1 implies expr2

- Examples :

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

context Person
inv: account->notEmpty() implies bank->notEmpty()
```

Variables

- The variables can be used to improve the comprehension of complex constraints

- Syntax : `let … in …`

```
context Person
inv: let money = account.balance->sum() in
age >= 18 implies money > 0
```

- To make the variables accessible everywhere : `def`

```
context Person
def: money : Integer = account.balance->sum()

context Person
inv: age >= 18 implies money > 0
```