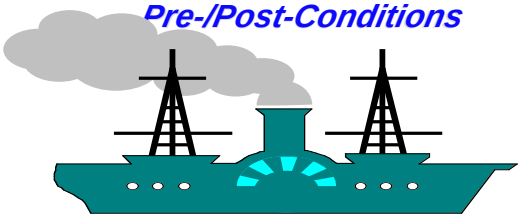
XIII. A Constraint Language for UML

The Constraint Language (CL)
Sets and Bags
Selectors and Invariants
Examples
Pre-/Post-Conditions



The Constraint Language

- *Some constraints can be adequately expressed graphically (e.g., multiplicity of an association).*
- *Some can not. For example, constraints within operation specifications (pre/post-conditions)*
- *The **Object Constraint Language** (OCL) [Warmer99] is a formal language for specifying constraints for UML class diagrams.*
- *We present a syntactic variant of a subset of OCL; let's call it **Constraint Language** (CL, for short.)*

Objects, Bags and Sets

- Objects are instances of classes, including predefined classes *Integer*, *Number* and *String*.
- Bags include zero or more objects and/or sets and/or other bags, possibly with duplicates, and no assumed order

  e.g., *{tom, maria, tom, sara, maria},{tom,{maria,tom},{}}*
- Two bags are equal iff they have the same number of the same elements:
  - ✓ *{tom, maria, tom, sara, maria} ≠ {tom,maria,sara}*
- Sets are bags with no duplicates.

CL Expressions

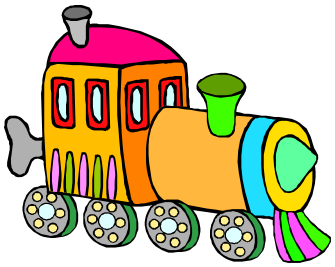
- CL expressions define ***constraints*** (or ***invariants***) for classes, which must be try for all their instances
  - ✓ e.g., "every employee earns less than his CEO"
- CL expressions also define conditions that must be true before an operation can be executed (***preconditions***) and conditions that must be true after (***postconditions***)
  - ✓ e.g., "Before withdrawCash(acct,amount), it must be that acct.balance ≥ amount" (precondition)
  - ✓ Or, "After withdrawCash(acct,amount) is executed, it must be that

    acct.balance(new) = acct.balance(old) - amount"

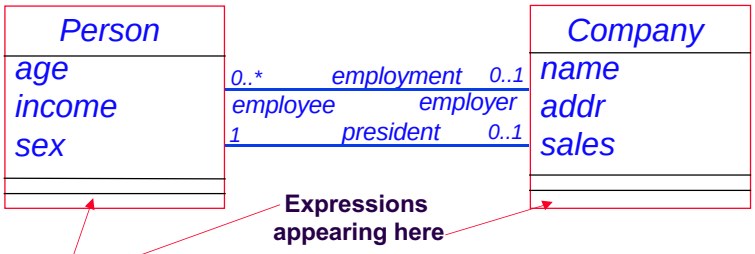
    (postcondition)

Contexts for CL Expressions

- *Every expression has a **context** which is the class within which it is defined.*
- *The special identifier self refers to an instance of the class within which it appears.*
- *The most basic CL expressions are called selectors and they return an object or a bag.*



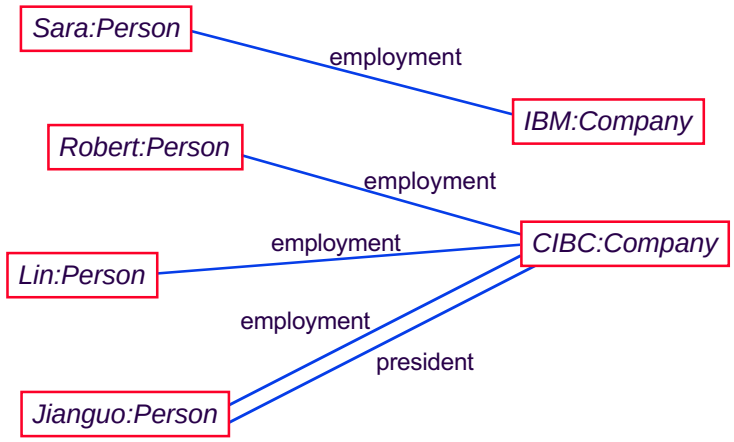
Selectors in Action



- ✓ *self.addr (or just addr) -- returns addr of a particular company;*
- ✓ *self.employment -- returns the set of all employees;*
- ✓ *self.employee -- returns the set of all employees as well;*
- ✓ *self.president -- returns the singleton set of presidents;*
- ✓ *self.employment -- set of all employers of a person;*
- ✓ *self.employer -- set of all employers of a person.*

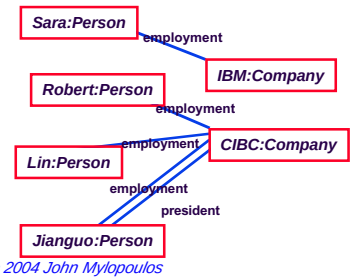
nil and empty

- When an attribute attr has no value for object obj, then *obj.attr* returns *nil* (no value).
- When there are no associated objects to an an object obj through association assoc (or role rl), then *obj.assoc* and *obj.rl* return the empty bag *{}* or *empty*.
- Note, nil ≠ {}.
- Moreover, {nil} = {}, {Sara,nil, nil} = {Sara} etc.
- This means that if Sara.age = nil, George.age = nil, then {Sara.age,George.age} = {}



Associations are Sets of Tuples

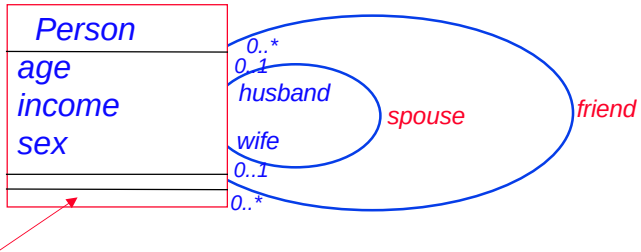
- You can think of associations as sets (no duplicates!) of tuples.
  - ✓ *Sara.employment* = {IBM}
  - ✓ *CIBC.employment* = {Robert,Lin,Jianguo}
  - ✓ *CIBC.employee* = {Robert,Lin,Jianguo}
  - ✓ *CIBC.employer* -- syntax error!



employment
(Sara,IBM)
(Robert,CIBC)
(Lin,CIBC)
(Jianguo,CIBC)
(Lin,IBC) X

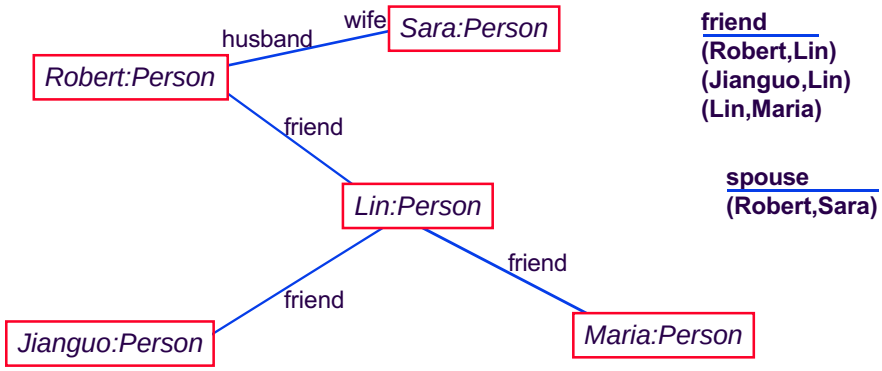
president
(Jianguo,CIBC)

Selectors for Symmetric Associations



- ✓ *self.age* (or just *age*) -- returns the age of a particular person;
- ✓ *self.husband* -- returns the set of all husbands;
- ✓ *self.wife* -- returns the set of all wives;
- ✓ *self.spouse* -- returns the set of all husbands and wives;
- ✓ *self.friend* -- returns the set of all friends.

- ✓ *Sara.spouse = {Robert}*
- ✓ *Lin.friend = {Robert,Jianguo,Maria}*
- ✓ *Maria.friend = {Lin}*



Applying Selectors to Bags

- *bag.attrName* = $\bigcup_{\text{obj} \in \text{bag}}$ *obj.attrName*
- *bag.assocName* = $\bigcup_{\text{obj} \in \text{bag}}$ *obj.assocName*
- *bag.roleName* = $\bigcup_{\text{obj} \in \text{bag}}$ *obj.roleName*
- *For example, suppose*
  - ✓ *Sara.friend = {Robert,Lin}*
  - ✓ *Jianguo.friend = {Robert,Maria}*
  - ✓ *{Sara,Jianguo}.friend = {Robert,Lin,Robert,Maria}*
  - ✓ *{Sara,Jianguo,Sara}.friend = {Robert,Lin,Robert,Maria,Robert,Lin}*

Composition of Selectors

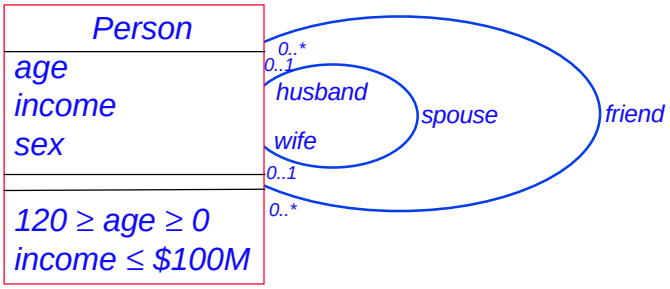
- Selectors can be composed:
  - ✓ *self.sel1.sel2 ≡ (self.sel1).sel2* means that we take the value of *self.sel1* (either a single value or a bag) and we apply to it *sel2.*
- For example, self.friend.income returns the bag of all income values of objects in the bag self.friend

  e.g., if *self.friend* = {Tom,Maria,Sara} and their incomes are respectively $16K, $19K and $16K, then

  self.friend.income = {$16K,$19K,$16K}

Constraints

- ***Constraints*** (or, ***invariants***) describe properties that must hold true for all the instances of the class.



- But also:
  - ✓ *(not empty(wife)) implies wife.sex = {female}*
  - ✓ *not empty(husband) implies husband.sex = {male}*

More Invariants

- *"If x is the wife of y, then y is the husband of x"*
  notEmpty(wife) implies {self} = self.wife.husband

or

Forall y[includes(self.wife,y)
implies includes(y.husband, self)]

- *"The president of a company is also its employee"*
  includes(self.employee,self.president)

...More...

- *"Popular persons have more than 50 friends"*
  - ✓ *We define a subclass of Person called PopularP and associate with it the invariant size(friend) > 50*
- *"For old rich persons, all their friends who are over 50 earn at least $100K"*
  - ✓ *We define a subclass of Person called OldRichP and associate with it the invariant*

  Forall y[(includes(friend,y) and y.age > 50)
  implies y.income ≥ $100K]

  Or, *Forall y[includes(select(friend,age>50),y)*
  implies y.income ≥ $100K]

  Or, *empty(select(select(friend,age>50),income<$100K))*

Bag Operations

size(bag) - returns the size (cardinality) of the bag

set(bag) - set that includes all elements of bag, no duplicates

sum(bag) - sum of elements in the bag (assumed numbers)

average(bag) - average of the bag

min(bag)/max(bag) - minimum/maximum element of the bag

empty(bag) - true if the bag is empty

includes(bag,object) - true if bag includes object

union(bag,bag) - union of two bags

intersection(bag,bag) - intersection of two bags

select(bag,predicate) - returns the subbag of bag whose elements satisfy the predicate

CL Expressions

- CL Expressions that define constraints, pre/post-conditions can now be defined as follows:
  - ✓ Boolean expressions using bag and object operations are CL Expressions;
  - ✓ If A, B are CL Expressions, then so are:
    - ✓ (A and B);
    - ✓ (A or B);
    - ✓ (not A);
    - ✓ (A implies B);
    - ✓ (Forall var) A;
    - ✓ (Exists var) A.
- Nothing else is a CL expression.

Another Example

- Suppose University class has an association studies to the Student class, and self refers to a University:
  - ✓ *self.studies* is a set of students, no duplicates;
  - ✓ *self.studies.age* is a bag -- many students can have the same age;
  - ✓ *average(self.studies.age)* returns the average age of all the students of a particular university;
  - ✓ *set(self.studies.degree)* returns the set of all degrees studied for in a university -- no duplicates!;

Pre- and Post-conditions in CL

- Pre-condition and post-condition expressions are associated to an operation/method and they describe
  - ✓ What must be true before the operation is executed (pre-condition);
  - ✓ What will be true once the operation is executed (post-condition).
- For example, we may want to say:

  Person::marryWife(p:Person)

  pre: self.wife = empty (not nil!)

  post: self.wife = {p}

More Examples

- *"When a person is promoted, her income is increased by at least 10%":*

  Person::promote(inc:DollarV)

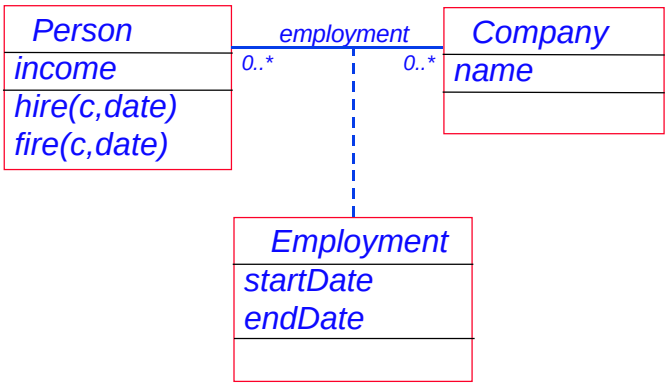
  pre: true

  *post: income ≥ income@pre * 1.1*

*The value of income **before** the operation*

Selecting Instances of Association Classes

- *Suppose we now want to keep track of a person's employments:*



Hiring and Firing

- *Person::hire(c:Company,d:Date)*
  pre: not includes(c.employment,self)
  post: includes(c.employment,self)
  and (self,c).startDate = d
- *Person::fire(c:Company,d:Date)*
  pre: includes(c.employment,self)
  and isBefore(startDate,d)
  post: (self,c).endDate = d

Selects a particular instance of the Employment association class

Additional Readings

- *[Warmer99]* Warmer, J. Kleppe, A. *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley 1999.
- *http://dec.bournemouth.ac.uk/dec_ind/swebster/UML_OCL/index.htm*