How to Check UML and OCL Models with USE

Martin Gogolla

July 2010

Abstract: This note contains practical hints on how to define UML and OCL models and how to check such models with the tool USE (UML-based Specification Environment) developed at the University of Bremen. One should obtain a link to the tool USE by searching Google with the terms 'use ocl bremen'. This note explains most of the USE functionality by a simple example describing civil status properties of persons.

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

1. WHAT IS CONTAINED IN THE FILES?

In order to work with the civil status example easily, all files should be placed in the directory 'c:/use/civstat/'. In your directory 'c:/use/civstat/' there should be 56 files which are partitioned into 9 groups.

```
------------------------------------------ files documenting this note --
- 00readme.pdf
- 00readme.sxw
- 00readme.txt
------------------------------------------ files for the abcd scenario --
- abcd_assl.cmd
- abcd_assl.olt
+ abcd_assl.pro
- abcd_cmd.cmd
- abcd_cmd.olt
+ abcd_cmd.pro
+ abcd_seqDiaDetail1.gif
+ abcd_seqDiaDetail2.gif
+ abcd_useScreenshot.gif
---------------------------------------- files for the bigamy scenario --
- bigamy.cmd
- bigamy.invs
+ bigamy.pro
-------------------------------------- central files showing the model --
+ civstat.assl
+ civstat.use
+ civstat_classDia.gif
----------------------------------------- files for the crowd scenario --
- crowd.cmd
- crowd.olt
+ crowd.pro
+ crowd_classExtent.gif
+ crowd_useScreenshot.gif
----------------------------------- papers explaining underlying ideas --
- Gogolla_2005_SOSYM.pdf
- Gogolla_2005_SOSYM.ps
- Gogolla_2007_SCP.pdf
- Gogolla_2007_SCP.ps
- Gogolla_2009_TAP.pdf
- Gogolla_2009_TAP.ps
```

```
------------------------------- files for the independence scenario --
- independence_allInvariants.gif
- independence_attributesDefined.cmd
+ independence_attributesDefined.gif
- independence_attributesDefined.pro
- independence_femaleHasNoWife.cmd
+ independence_femaleHasNoWife.gif
- independence_femaleHasNoWife.pro
- independence_maleHasNoHusband.cmd
+ independence_maleHasNoHusband.gif
- independence_maleHasNoHusband.pro
- independence_nameCapitalThenSmallLetters.cmd
+ independence_nameCapitalThenSmallLetters.gif
- independence_nameCapitalThenSmallLetters.pro
- independence_nameIsUnique.cmd
+ independence_nameIsUnique.gif
- independence_nameIsUnique.pro
------------------------------- cmd files defining model operations --
+ Person_birth.cmd
+ Person_death_married.cmd
+ Person_death_unmarried.cmd
+ Person_divorce.cmd
+ Person_marry.cmd
-------------------------------------- files for the world scenario --
- world.cmd
- world.olt
+ world.pro
+ world_objDia.gif
- world_query.cmd
+ world_query.pro
-----------------------------------------------------------------------
```

```
Files ending with 'use', 'assl', 'cmd', 'invs', or 'olt' can be
processed by the USE tool. All files with other suffixes ('pdf',
'sxw', 'txt', 'pro', 'gif', or 'ps') are the result of documenting the
work ('gif', 'pro', ...) with USE or explain basic ideas ('ps',
'pdf'). The file suffixes indicate the file content as follows.

use  - definition of the UML and OCL model (classes, invariants, etc.)
assl - definition of parametrized scripts for manipulating the model
cmd  - scripts or script calls realizing scenarios
invs - additional invariants to be loaded dynamically
olt  - object diagram layout

pdf  - pdf file for documentation
txt  - plain text files for documentation
pro  - protocol file (plain text) documenting execution of a cmd file
gif  - graphic file containing a UML diagram or something similar
ps   - ps file for documentation

All files enumerated above with '+' are shown in the file
'00readme.pdf'.
```

++

2. HOW CAN I RUN USE AND EXECUTE SCENARIOS?

Download, install and start USE. Consult the USE documentation if needed. USE will show up with a GUI window and a COMMAND LINE window. Open the file 'c:/use/civstat/abcd_cmd.cmd' with a text editor. The first line of that file looks as follows:

```
-- read c:/use/civstat/abcd_cmd.cmd
   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

A cmd file contains a sequence of commands which can be processed by USE. The characters '--' indicate that the rest of the line is a comment. Copy that part of this first line into your clipboard which is marked above with the character '^'. Paste the command which you have copied from your clipboard into the COMMAND LINE window and press return. USE will execute all commands in the file 'c:/use/civstat/abcd_cmd.cmd' and will build up an object diagram. USE has constructed several object diagrams before reaching the final one. All executed operations can be traced in the sequence diagram view.

Execution of all above mentioned scenarios can be done in the same way as for the first scenario, i.e., by loading the respective cmd file into a text editor, by copying from the first line the read command together with the respective file name and by pasting this command into the COMMAND LINE window. This will work for the following cmd files:

```
abcd_assl.{cmd|pro}
abcd_cmd.{cmd|pro}
bigamy.{cmd|pro}
crowd.{cmd|pro}
independence_attributesDefined.{cmd|pro}
independence_femaleHasNoWife.{cmd|pro}
independence_maleHasNoHusband.{cmd|pro}
independence_nameCapitalThenSmallLetters.{cmd|pro}
independence_nameIsUnique.{cmd|pro}
world.{cmd|pro}
world_query.{cmd|pro}
```

These scenarios are documented with a protocol file having the same name as the cmd file but possessing the suffix pro. These pro files are ordinary text files. They are edited versions (long files names are shortened) of the output given by USE in the COMMAND LINE window.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

3. WHAT DO THE DIFFERENT SCENARIOS SHOW?

Scenario 'abcd': A simple object diagram with four persons (ada, bob,
   cyd, dan) is built up. The persons are created, partly get married or
   divorced or die. This scenario is expressed in two versions, namely
   one version realizing the operations with cmd files (abcd_cmd.cmd)
   and one version realizing the operations using the assl file
   (abcd_assl.cmd).

Scenario 'bigamy': This scenario (bigamy.cmd) checks whether a bigamy
   situation can be constructed under the stated invariants. A large
   number of possible bigamy situations is tried with the script
   attemptBigamy() from the civstat.assl file, but none is found. This
   proves that at least in the enumerated search space, the UML and OCL
   model is bigamy free.

Scenario 'crowd': The cmd file crowd.cmd builds up an invalid object
   diagram where one model-inherent multiplicity constraint and two
   explicit invariants are violated. This scenario may be regarded as a
   negative test case which has to lead to a constraint violation.

Scenario 'independence': This scenario considers the relationship
   between the five invariants and consists of five cmd files
   (independence_attributesDefined.cmd,
   independence_femaleHasNoWife.cmd, independence_maleHasNoHusband.cmd,
   independence_nameCapitalThenSmallLetters.cmd,
   independence_nameIsUnique.cmd). It builds up five object diagrams
   where each single object diagram violates exactly one invariant and
   satisfies the other four invariants. The construction of these five
   object diagrams proves that the five invariants are independent from
   each other, i.e., no single invariant is a consequence of the other
   invariants. In other words, the five invariants are non-redundant and
   express independent requirements.

Scenario 'world': Here a larger object diagram is constructed with the
   cmd file world.cmd that shows that the invariants are consistent,
   i.e., the invariants are not contradictory to each other. The object
   diagram is constructed by the script world() which is parametrized
   with the number of expected female and male persons and the number of
   expected mariages. Person attributes like name or gender are filled
   with values representing real-world situations.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

++

A. Central Files and cmd Files for Operations

++

civstat.use

```
model CivilStatusWorld

enum CivilStatus {single, married, divorced, widowed}
enum Gender {female, male}

class Person

attributes
  name:String
  civstat:CivilStatus
  gender:Gender
  alive:Boolean

operations

birth(aName:String, aGender:Gender)
pre  freshUnlinkedPerson: name.isUndefined and civstat.isUndefined and
     gender.isUndefined and alive.isUndefined and
     wife.isUndefined and husband.isUndefined
post nameAssigned: name=aName -- equivalent to 'aName=self.name'
post civstatAssigned: civstat=#single
post genderAssigned: gender=aGender
post isAliveAssigned: alive=true -- equivalent to 'alive'

marry(aSpouse:Person)
pre  aSpouseDefined: aSpouse.isDefined
pre  isAlive: alive
pre  aSpouseAlive: aSpouse.alive
pre  isUnmarried: civstat<>#married
pre  aSpouseUnmarried: aSpouse.civstat<>#married
pre  differentGenders: gender<>aSpouse.gender
post isMarried: civstat=#married
post femaleHasMarriedHusband: gender=#female implies
     husband=aSpouse and husband.civstat=#married
post maleHasMarriedWife: gender=#male implies
     wife=aSpouse and wife.civstat=#married

divorce()
pre  isMarried: civstat=#married
pre  isAlive: alive
pre  husbandAlive: gender=#female implies husband.alive
pre  wifeAlive: gender=#male implies wife.alive
post isDivorced: civstat=#divorced
post husbandDivorced: gender=#female implies
     husband.isUndefined and husband@pre.civstat=#divorced
post wifeDivorced: gender=#male implies
     wife.isUndefined and wife@pre.civstat=#divorced
```

```
death()
pre  isAlive: alive
post notAlive: not(alive)
post husbandWidowed: gender=#female and husband@pre.isDefined implies
     husband@pre.wife.isUndefined and husband@pre.civstat=#widowed
post wifeWidowed: gender=#male and wife@pre.isDefined implies
     wife@pre.husband.isUndefined and wife@pre.civstat=#widowed

spouse():Person=if gender=#female then husband else wife endif

constraints -- invariants can be defined also outside the class see (*)
  inv attributesDefined: name.isDefined and civstat.isDefined and
      gender.isDefined and alive.isDefined
  inv nameCapitalThenSmallLetters: -- name.matches('[A-Z][a-z]*')
      let small:Set(String)=
        Set{'a','b','c','d','e','f','g','h','i','j','k','l','m',
            'n','o','p','q','r','s','t','u','v','w','x','y','z'} in
      let capital:Set(String)=
        Set{'A','B','C','D','E','F','G','H','I','J','K','L','M',
            'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in
      capital->includes(name.substring(1,1)) and
      Set{2..name.size}->forAll(i |
        small->includes(name.substring(i,i)))
  inv nameIsUnique: Person.allInstances->forAll(self2|
      self<>self2 implies self.name<>self2.name)
  inv femaleHasNoWife: gender=#female implies wife.isUndefined
  inv maleHasNoHusband: gender=#male implies husband.isUndefined

end

association Marriage between
  Person [0..1] role wife
  Person [0..1] role husband
end

-- constraints -- (*)
--
-- context Person inv attributesDefined2:
--   name.isDefined and civstat.isDefined and
--   gender.isDefined and alive.isDefined

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Person_birth.cmd

```
-- Person::birth(aName:String,aGender:Gender)
!set self.name:=aName
!set self.civstat:=#single
!set self.gender:=aGender
!set self.alive:=true
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Person_marry.cmd

```
-- Person::marry(aSpouse:Person)
!set self.civstat:=#married
!set aSpouse.civstat:=#married
!insert (if self.gender=#female then self else aSpouse endif,
         if self.gender=#female then aSpouse else self endif)
        into Marriage
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Person_divorce.cmd

```
-- Person::divorce()
!set self.civstat:=#divorced
!set self.spouse().civstat:=#divorced
!delete (if self.gender=#female then self else self.wife endif,
         if self.gender=#female then self.husband else self endif)
        from Marriage
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Person_death_married.cmd

```
-- Person::death() -- for married Person objects
!set self.alive:=false
!set self.spouse().civstat:=#widowed
!delete (if self.gender=#female then self else self.wife endif,
         if self.gender=#female then self.husband else self endif)
        from Marriage
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Person_death_unmarried.cmd

```
-- Person::death() -- for unmarried Person objects
!set self.alive:=false
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

civstat.assl

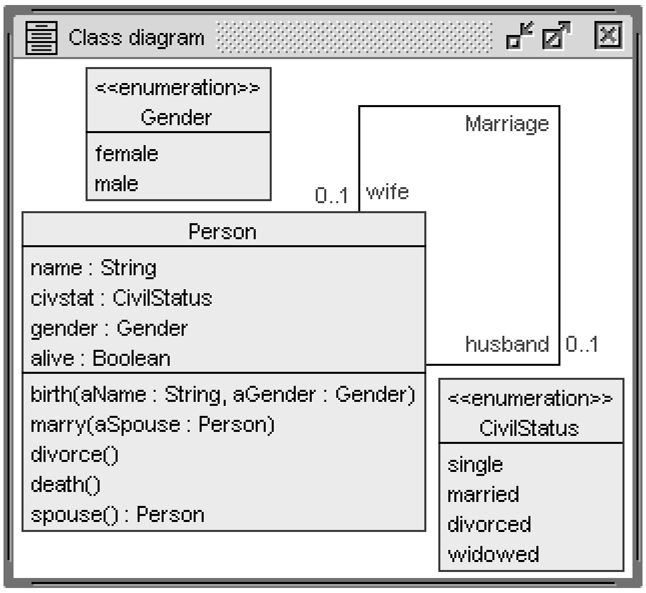
```
------------------------------------------------------------------------

procedure Person_birth(self:Person,aName:String,aGender:Gender)
begin
[self].name:=[aName];
[self].civstat:=[#single];
[self].gender:=[aGender];
[self].alive:=[true];
end;

procedure Person_marry(self:Person,aSpouse:Person)
```

```
begin
[self].civstat:=[#married]; [aSpouse].civstat:=[#married];
if [self.gender=#female] then
  begin Insert(Marriage,[self],[aSpouse]); end
else -- [self.gender=#male]
  begin Insert(Marriage,[aSpouse],[self]); end;
end;

procedure Person_divorce(self:Person)
begin
[self].civstat:=[#divorced];
if [self.gender=#female] then
  begin [self.husband].civstat:=[#divorced];
  Delete(Marriage,[self],[self.husband]); end
else -- [self.gender=#male]
  begin [self.wife].civstat:=[#divorced];
  Delete(Marriage,[self.wife],[self]); end;
end;

procedure Person_death(self:Person)
begin
[self].alive:=[false];
if [self.husband.isDefined] then -- [self.gender=#female]
  begin [self.husband].civstat:=[#widowed];
  Delete(Marriage,[self],[self.husband]); end;
if [self.wife.isDefined] then -- [self.gender=#male]
  begin [self.wife].civstat:=[#widowed];
  Delete(Marriage,[self.wife],[self]); end;
end;

------------------------------------------------------------------------

procedure crowd(numFemale:Integer, numMale:Integer, numMarriage:Integer)
var theFemales: Sequence(Person), theMales: Sequence(Person),
    f: Person, m: Person;
begin
theFemales:=CreateN(Person,[numFemale]);
theMales:=CreateN(Person,[numMale]);
for i:Integer in [Sequence{1..numFemale}]
  begin [theFemales->at(i)].name:=Any([Sequence{'Ada','Bel','Cam','Day',
    'Eva','Flo','Gen','Hao','Ina','Jen'}]);
  [theFemales->at(i)].civstat:=
    Any([Sequence{#single,#married,#divorced,#widowed}]);
  [theFemales->at(i)].gender:=Any([Sequence{#female,#male}]);
  [theFemales->at(i)].alive:=Any([Sequence{false,true}]); end;
for i:Integer in [Sequence{1..numMale}]
  begin [theMales->at(i)].name:=Any([Sequence{'Ali','Bob','Cyd','Dan',
    'Eli','Fox','Gil','Hal','Ike','Jan'}]);
  [theMales->at(i)].civstat:=
    Any([Sequence{#single,#married,#divorced,#widowed}]);
  [theMales->at(i)].gender:=Any([Sequence{#female,#male}]);
  [theMales->at(i)].alive:=Any([Sequence{false,true}]); end;
```

```
for i:Integer in [Sequence{1..numMarriage}]
  begin f:=Any([theFemales]); m:=Any([theMales]);
  Insert(Marriage,[f],[m]); end;
end;

--------------------------------------------------------------------------

procedure world(numFemale:Integer, numMale:Integer, numMarriage:Integer)
-- numMarriage<=numFemale<=26, numMarriage<=numMale<=26
var theFemales: Sequence(Person), theMales: Sequence(Person),
    f: Person, m: Person;
begin
theFemales:=CreateN(Person,[numFemale]);
theMales:=CreateN(Person,[numMale]);
for i:Integer in [Sequence{1..numFemale}]
  begin [theFemales->at(i)].name:=Any([Sequence{'Ada','Bel','Cam','Day',
     'Eva','Flo','Gen','Hao','Ina','Jen','Kia','Lan','Mae','Nan','Oki',
     'Pam','Quao','Rae','Sen','Tip','Una','Vea','Wan','Xia','Yan','Zoe'}
    ->reject(n|Person.allInstances->exists(p|p.name=n))]);
  [theFemales->at(i)].civstat:=[#single];
  [theFemales->at(i)].gender:=[#female];
  [theFemales->at(i)].alive:=[true]; end;
for i:Integer in [Sequence{1..numMale}]
  begin [theMales->at(i)].name:=Any([Sequence{'Ali','Bob','Cyd','Dan',
    'Eli','Fox','Gil','Hal','Ike','Jan','Kim','Leo','Max','Nam','Ole',
    'Pat','Quin','Rex','Sam','Tom','Ulf','Vic','Wei','Xan','Yul','Zan'}
    ->reject(n|Person.allInstances->exists(p|p.name=n))]);
  [theMales->at(i)].civstat:=[#single];
  [theMales->at(i)].gender:=[#male];
  [theMales->at(i)].alive:=[true]; end;
for i:Integer in [Sequence{1..numMarriage}]
  begin f:=Any([theFemales->reject(p|p.husband.isDefined)]);
  m:=Any([theMales->reject(p|p.wife.isDefined)]);
  [f].civstat:=[#married]; [m].civstat:=[#married];
  Insert(Marriage,[f],[m]); end;
end;

--------------------------------------------------------------------------

procedure attemptBigamy()
var p: Person, w: Person, h:Person, thePersons: Sequence(Person);
--   w -wife-----husband- p -wife-----husband- h
begin
thePersons:=CreateN(Person,[3]);
for i:Integer in [Sequence{1..3}]
  begin [thePersons->at(i)].name:=Try([Sequence{'A','B','C'}]);
  [thePersons->at(i)].civstat:=
    Try([Sequence{#single,#married,#divorced,#widowed}]);
  [thePersons->at(i)].gender:=Try([Sequence{#female,#male}]);
  [thePersons->at(i)].alive:=Try([Sequence{false,true}]); end;
p:=Try([thePersons]);
w:=Try([thePersons->excluding(p)]);
h:=Try([thePersons->excluding(p)->excluding(w)]);
Insert(Marriage,[w],[p]); Insert(Marriage,[p],[h]);
end; -- [(3*4*2*2)^3]*(3*2*1) = [48^3]*6 = 110552*6 = 663552
```

++

classDia.gif

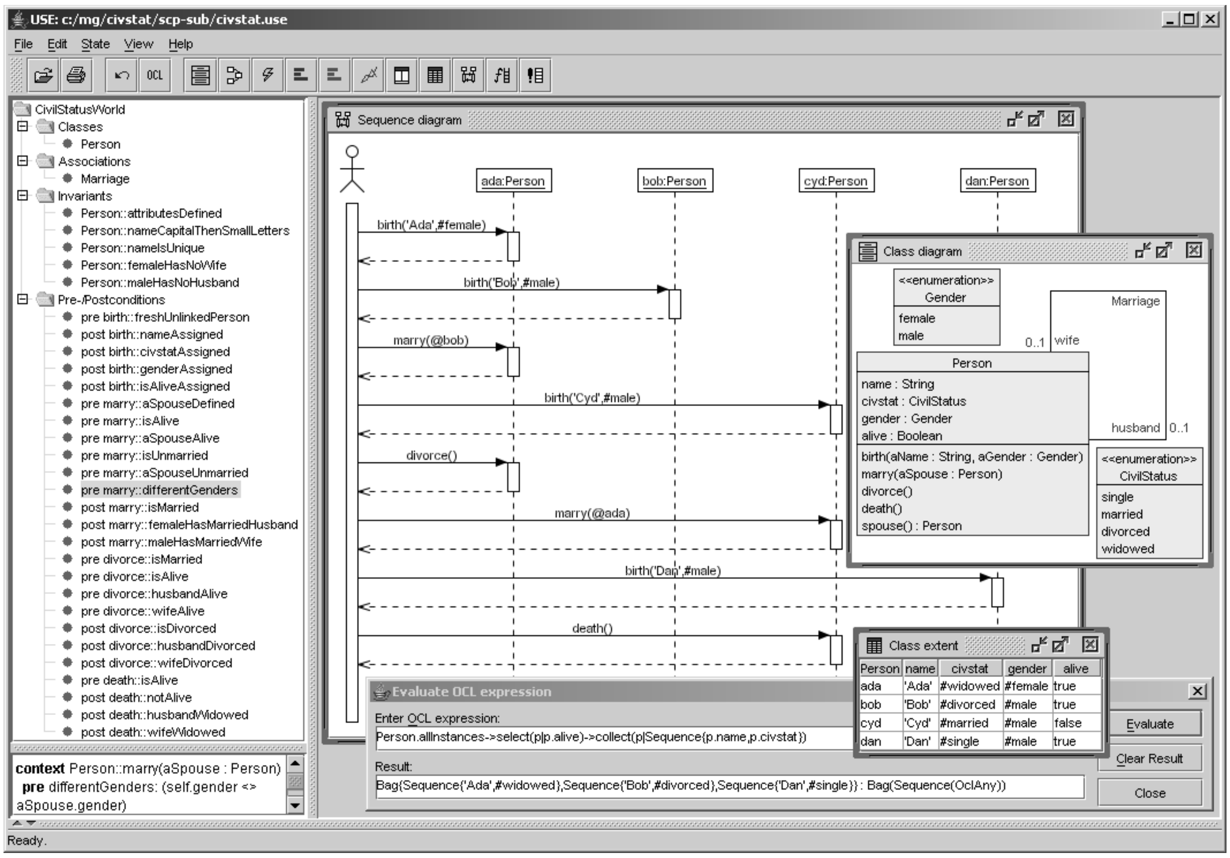


++

B. Scenario abcd (ada, bob, cyd, dan)

++

abcd_useScreenshot.gif



```
CivilStatusWorld
  Classes
    Person
  Associations
    Marriage
  Invariants
    Person::attributesDefined
    Person::nameCapitalThenSmallLetters
    Person::nameIsUnique
    Person::femaleHasNoWife
    Person::maleHasNoHusband
  Pre-/Postconditions
    pre birth::freshUnlinkedPerson
    post birth::nameAssigned
    post birth::civstatAssigned
    post birth::genderAssigned
    post birth::isAliveAssigned
    pre marry::aSpouseDefined
    pre marry::isAlive
    pre marry::aSpouseAlive
    pre marry::isUnmarried
    pre marry::aSpouseUnmarried
    pre marry::differentGenders
    post marry::isMarried
    post marry::femaleHasMarriedHusband
    post marry::maleHasMarriedWife
    pre divorce::isMarried
    pre divorce::isAlive
    pre divorce::husbandAlive
    pre divorce::wifeAlive
    post divorce::isDivorced
    post divorce::husbandDivorced
    post divorce::wifeDivorced
    pre death::isAlive
    post death::notAlive
    post death::husbandWidowed
    post death::wifeWidowed
```

```
context Person::marry(aSpouse : Person)
 pre differentGenders: (self.gender <>
aSpouse.gender)
```

Person	name	civstat	gender	alive
ada	'Ada'	#widowed	#female	true
bob	'Bob'	#divorced	#male	true
cyd	'Cyd'	#married	#male	false
dan	'Dan'	#single	#male	true

Enter OCL expression:

```
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})
```

Result:

```
Bag{Sequence{'Ada',#widowed},Sequence{'Bob',#divorced},Sequence{'Dan',#single}} : Bag(Sequence(OclAny))
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

abcd_cmd.pro

```
use> open civstat.use

use> !create ada:Person

use> !openter ada birth('Ada',#female)
     precondition `freshUnlinkedPerson' is true
use> read Person_birth.cmd
     Person_birth.cmd> -- Person::birth(aName:String,aGender:Gender)
     Person_birth.cmd> !set self.name:=aName
     Person_birth.cmd> !set self.civstat:=#single
     Person_birth.cmd> !set self.gender:=aGender
     Person_birth.cmd> !set self.alive:=true

use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !create bob:Person

use> !openter bob birth('Bob',#male)
     precondition `freshUnlinkedPerson' is true
use> read Person_birth.cmd
     Person_birth.cmd> -- Person::birth(aName:String,aGender:Gender)
     Person_birth.cmd> !set self.name:=aName
     Person_birth.cmd> !set self.civstat:=#single
     Person_birth.cmd> !set self.gender:=aGender
     Person_birth.cmd> !set self.alive:=true

use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter ada marry(bob)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> read Person_marry.cmd
     Person_marry.cmd> -- Person::marry(aSpouse:Person)
     Person_marry.cmd> !set self.civstat:=#married
     Person_marry.cmd> !set aSpouse.civstat:=#married
     Person_marry.cmd> !insert
     (if self.gender=#female then self else aSpouse endif,
      if self.gender=#female then aSpouse else self endif) into Marriage
```

```
use> !opexit
     postcondition `isMarried' is true
     postcondition `femaleHasMarriedHusband' is true
     postcondition `maleHasMarriedWife' is true

use> !create cyd:Person

use> !openter cyd birth('Cyd',#male)
     precondition `freshUnlinkedPerson' is true
use> read Person_birth.cmd
     Person_birth.cmd> -- Person::birth(aName:String,aGender:Gender)
     Person_birth.cmd> !set self.name:=aName
     Person_birth.cmd> !set self.civstat:=#single
     Person_birth.cmd> !set self.gender:=aGender
     Person_birth.cmd> !set self.alive:=true

use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter ada divorce()
     precondition `isMarried' is true
     precondition `isAlive' is true
     precondition `husbandAlive' is true
     precondition `wifeAlive' is true
use> read Person_divorce.cmd
     Person_divorce.cmd> -- Person::divorce()
     Person_divorce.cmd> !set self.civstat:=#divorced
     Person_divorce.cmd> !set self.spouse().civstat:=#divorced
     Person_divorce.cmd> !delete
     (if self.gender=#female then self else self.wife endif,
      if self.gender=#female then self.husband else self endif)
     from Marriage

use> !opexit
     postcondition `isDivorced' is true
     postcondition `husbandDivorced' is true
     postcondition `wifeDivorced' is true

use> !openter cyd marry(ada)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> read Person_marry.cmd
     Person_marry.cmd> -- Person::marry(aSpouse:Person)
     Person_marry.cmd> !set self.civstat:=#married
     Person_marry.cmd> !set aSpouse.civstat:=#married
     Person_marry.cmd> !insert
     (if self.gender=#female then self else aSpouse endif,
      if self.gender=#female then aSpouse else self endif) into Marriage
```

```
use> !opexit
     postcondition `isMarried' is true
     postcondition `femaleHasMarriedHusband' is true
     postcondition `maleHasMarriedWife' is true

use> !create dan:Person

use> !openter dan birth('Dan',#male)
     precondition `freshUnlinkedPerson' is true
use> read Person_birth.cmd
     Person_birth.cmd> -- Person::birth(aName:String,aGender:Gender)
     Person_birth.cmd> !set self.name:=aName
     Person_birth.cmd> !set self.civstat:=#single
     Person_birth.cmd> !set self.gender:=aGender
     Person_birth.cmd> !set self.alive:=true

use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter cyd death()
     precondition `isAlive' is true
use> read Person_death_married.cmd
     Person_death_married.cmd> -- Person::death()
     Person_death_married.cmd> -- for married Person objects
     Person_death_married.cmd> !set self.alive:=false
     Person_death_married.cmd> !set self.spouse().civstat:=#widowed
     Person_death_married.cmd> !delete
     (if self.gender=#female then self else self.wife endif,
      if self.gender=#female then self.husband else self endif)
     from Marriage

use> !opexit
     postcondition `notAlive' is true
     postcondition `husbandWidowed' is true
     postcondition `wifeWidowed' is true

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

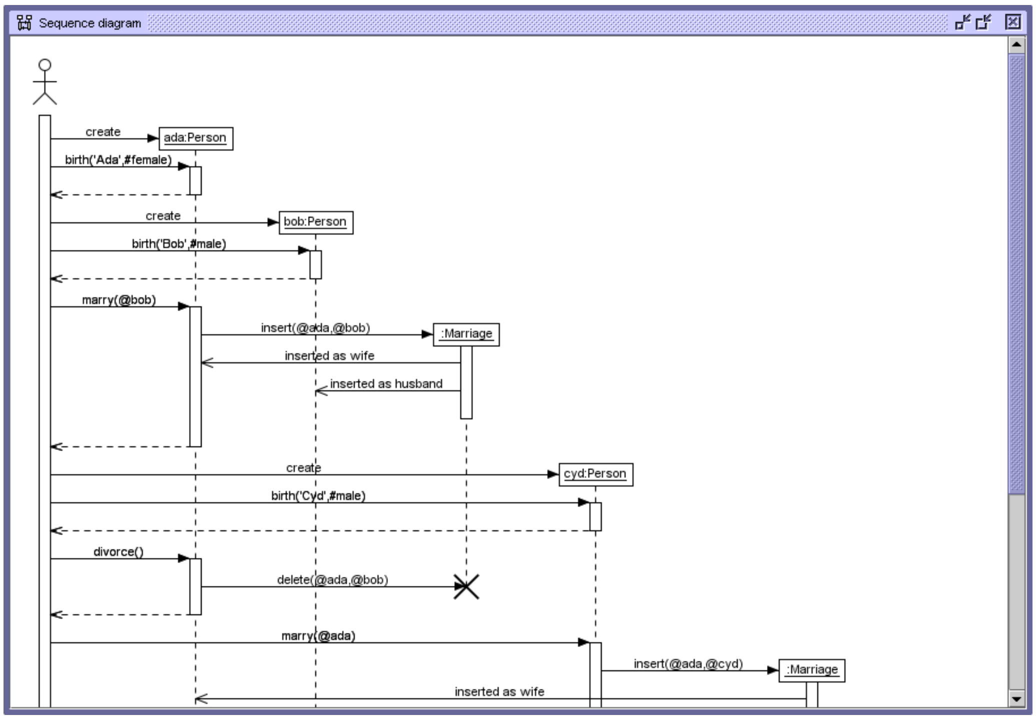
abcd_assl.pro

```
use> open civstat.use

use> !create ada:Person

use> !openter ada birth('Ada',#female)
     precondition `freshUnlinkedPerson' is true
use> gen start civstat.assl Person_birth(self,aName,aGender)
use> gen result
     Random number generator was initialized with 5871.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
```

```
     !set @ada.name := 'Ada'
     !set @ada.civstat := #single
     !set @ada.gender := #female
     !set @ada.alive := true
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !create bob:Person

use> !openter bob birth('Bob',#male)
     precondition `freshUnlinkedPerson' is true
use> gen start civstat.assl Person_birth(self,aName,aGender)
use> gen result
     Random number generator was initialized with 1308.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @bob.name := 'Bob'
     !set @bob.civstat := #single
     !set @bob.gender := #male
     !set @bob.alive := true
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter ada marry(bob)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> gen start civstat.assl Person_marry(self,aSpouse)
use> gen result
     Random number generator was initialized with 8047.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @ada.civstat := #married
     !set @bob.civstat := #married
     !insert (ada,bob) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `isMarried' is true
     postcondition `femaleHasMarriedHusband' is true
     postcondition `maleHasMarriedWife' is true

use> !create cyd:Person

use> !openter cyd birth('Cyd',#male)
     precondition `freshUnlinkedPerson' is true
use> gen start civstat.assl Person_birth(self,aName,aGender)
use> gen result
     Random number generator was initialized with 2430.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @cyd.name := 'Cyd'
     !set @cyd.civstat := #single
     !set @cyd.gender := #male
     !set @cyd.alive := true
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter ada divorce()
     precondition `isMarried' is true
     precondition `isAlive' is true
     precondition `husbandAlive' is true
     precondition `wifeAlive' is true
use> gen start civstat.assl Person_divorce(self)
use> gen result
     Random number generator was initialized with 3210.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @ada.civstat := #divorced
     !set @bob.civstat := #divorced
     !delete (ada,bob) from Marriage
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `isDivorced' is true
     postcondition `husbandDivorced' is true
     postcondition `wifeDivorced' is true

use> !openter cyd marry(ada)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> gen start civstat.assl Person_marry(self,aSpouse)
use> gen result
     Random number generator was initialized with 7593.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @cyd.civstat := #married
     !set @ada.civstat := #married
     !insert (ada,cyd) into Marriage
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `isMarried' is true
     postcondition `femaleHasMarriedHusband' is true
     postcondition `maleHasMarriedWife' is true

use> !create dan:Person

use> !openter dan birth('Dan',#male)
     precondition `freshUnlinkedPerson' is true
use> gen start civstat.assl Person_birth(self,aName,aGender)
use> gen result
     Random number generator was initialized with 3483.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @dan.name := 'Dan'
     !set @dan.civstat := #single
     !set @dan.gender := #male
     !set @dan.alive := true
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `nameAssigned' is true
     postcondition `civstatAssigned' is true
     postcondition `genderAssigned' is true
     postcondition `isAliveAssigned' is true

use> !openter cyd death()
     precondition `isAlive' is true
use> gen start civstat.assl Person_death(self)
use> gen result
     Random number generator was initialized with 3761.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !set @cyd.alive := false
     !set @ada.civstat := #widowed
     !delete (ada,cyd) from Marriage
use> gen result accept
     Generated result (system state) accepted.
use> !opexit
     postcondition `notAlive' is true
     postcondition `husbandWidowed' is true
     postcondition `wifeWidowed' is true
```

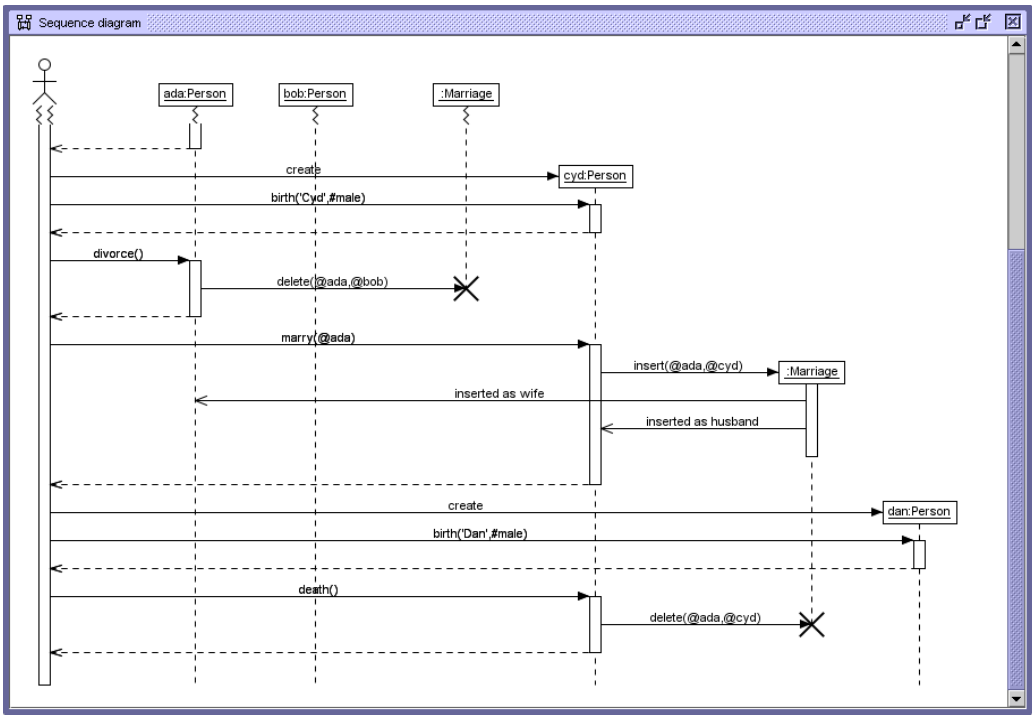
++

abcd_seqDiaDetail1.gif



++

abcd_seqDiaDetail2.gif



```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

C. Scenario bigamy

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

bigamy.pro

```
use> open civstat.use

-- File bigamy.invs contains:
-- context Person inv bigamy: Person.allInstances->exists(p|
--   p.wife.isDefined and p.husband.isDefined)

use> gen load bigamy.invs
     Added invariants: Person::bigamy
use> gen start civstat.assl attemptBigamy()
use> gen result
     Random number generator was initialized with 5649.
     Checked 663552 snapshots. Result: No valid state found.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

D. Scenario crowd

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

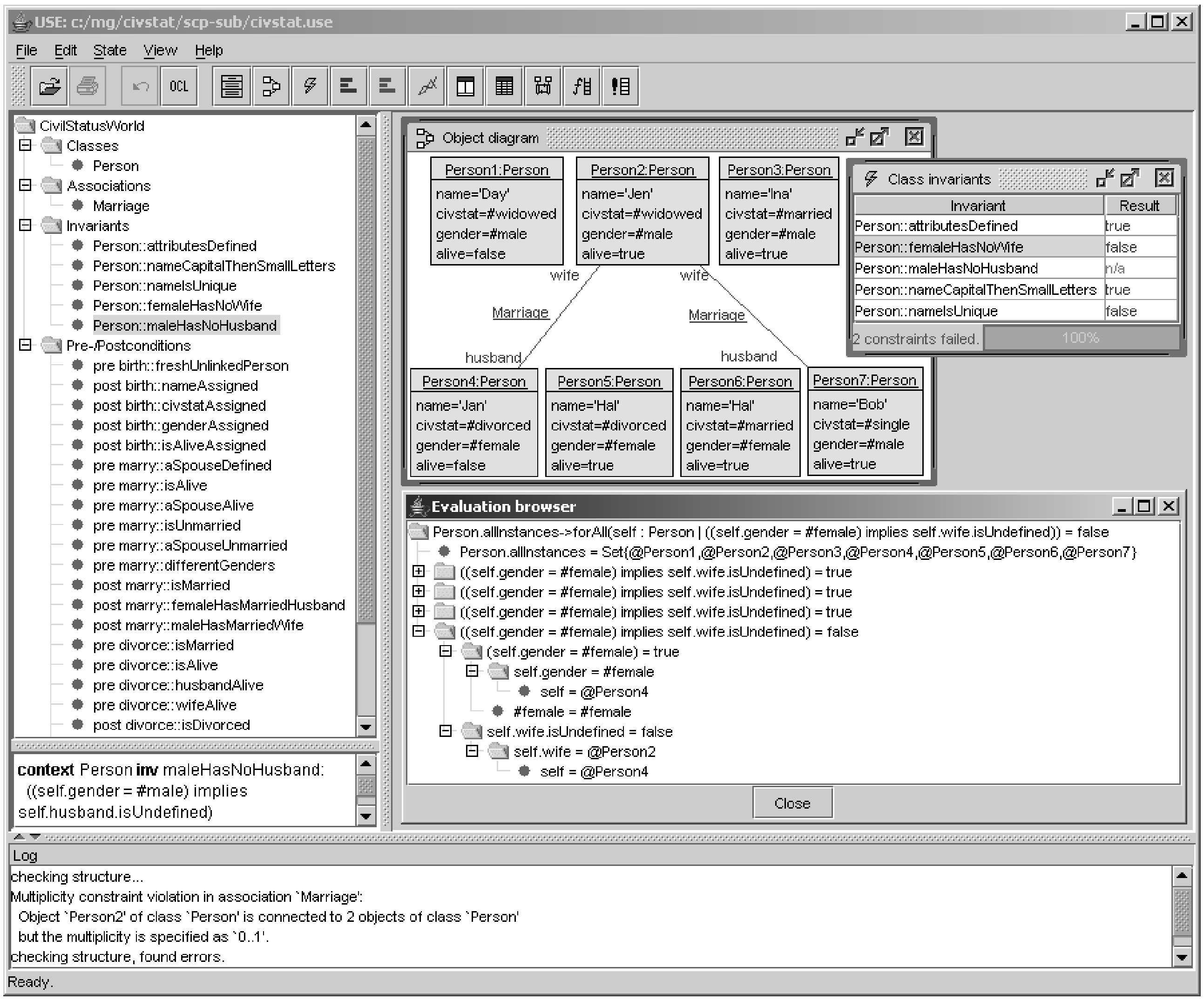
crowd.pro

```
use> open civstat.use

use> gen flags Person::attributesDefined +d
use> gen flags Person::femaleHasNoWife +d
use> gen flags Person::maleHasNoHusband +d
use> gen flags Person::nameCapitalThenSmallLetters +d
use> gen flags Person::nameIsUnique +d

use> gen start -s -r 2115 civstat.assl crowd(3,4,2)
use> gen result
     Random number generator was initialized with 2115.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !create Person4,Person5,Person6,Person7 : Person
     !set @Person1.name := 'Day'
     !set @Person1.civstat := #widowed
     !set @Person1.gender := #male
     !set @Person1.alive := false
     !set @Person2.name := 'Jen'
     !set @Person2.civstat := #widowed
     !set @Person2.gender := #male
     !set @Person2.alive := true
     !set @Person3.name := 'Ina'
     !set @Person3.civstat := #married
```

```
      !set @Person3.gender := #male
      !set @Person3.alive := true
      !set @Person4.name := 'Jan'
      !set @Person4.civstat := #divorced
      !set @Person4.gender := #female
      !set @Person4.alive := false
      !set @Person5.name := 'Hal'
      !set @Person5.civstat := #divorced
      !set @Person5.gender := #female
      !set @Person5.alive := true
      !set @Person6.name := 'Hal'
      !set @Person6.civstat := #married
      !set @Person6.gender := #female
      !set @Person6.alive := true
      !set @Person7.name := 'Bob'
      !set @Person7.civstat := #single
      !set @Person7.gender := #male
      !set @Person7.alive := true
      !insert (Person2,Person7) into Marriage
      !insert (Person2,Person4) into Marriage
use> gen result accept
      Generated result (system state) accepted.
use> check
      checking structure...
      Multiplicity constraint violation in association `Marriage': Object
        `Person2' of class `Person' is connected to 2 objects of class
        `Person' but the multiplicity is specified as `0..1'.
      checking invariants...
      checking invariant (1) `Person::attributesDefined': OK.
      checking invariant (2) `Person::femaleHasNoWife': FAILED.
      checking invariant (3) `Person::maleHasNoHusband': N/A
      checking invariant (4) `Person::nameCapitalThenSmallLetters': OK.
      checking invariant (5) `Person::nameIsUnique': FAILED.
      checked 5 invariants in 0.047s, 2 failures.
```

++

`crowd_useScreenshot.gif`



CivilStatusWorld
- Classes
  - Person
- Associations
  - Marriage
- Invariants
  - Person::attributesDefined
  - Person::nameCapitalThenSmallLetters
  - Person::nameIsUnique
  - Person::femaleHasNoWife
  - Person::maleHasNoHusband
- Pre-/Postconditions
  - pre birth::freshUnlinkedPerson
  - post birth::nameAssigned
  - post birth::civstatAssigned
  - post birth::genderAssigned
  - post birth::isAliveAssigned
  - pre marry::aSpouseDefined
  - pre marry::isAlive
  - pre marry::aSpouseAlive
  - pre marry::isUnmarried
  - pre marry::aSpouseUnmarried
  - pre marry::differentGenders
  - post marry::isMarried
  - post marry::femaleHasMarriedHusband
  - post marry::maleHasMarriedWife
  - pre divorce::isMarried
  - pre divorce::isAlive
  - pre divorce::husbandAlive
  - pre divorce::wifeAlive
  - post divorce::isDivorced

```
context Person inv maleHasNoHusband:
  ((self.gender = #male) implies
self.husband.isUndefined)
```

```
Person1:Person        Person2:Person        Person3:Person
name='Day'            name='Jen'            name='Ina'
civstat=#widowed      civstat=#widowed      civstat=#married
gender=#male          gender=#male          gender=#male
alive=false           alive=true            alive=true

wife   wife   Marriage   Marriage   husband   husband

Person4:Person        Person5:Person        Person6:Person        Person7:Person
name='Jan'            name='Hal'            name='Hal'            name='Bob'
civstat=#divorced     civstat=#divorced     civstat=#married      civstat=#single
gender=#female        gender=#female        gender=#female        gender=#male
alive=false           alive=true            alive=true            alive=true
```

Invariant	Result
Person::attributesDefined	true
Person::femaleHasNoWife	false
Person::maleHasNoHusband	n/a
Person::nameCapitalThenSmallLetters	true
Person::nameIsUnique	false

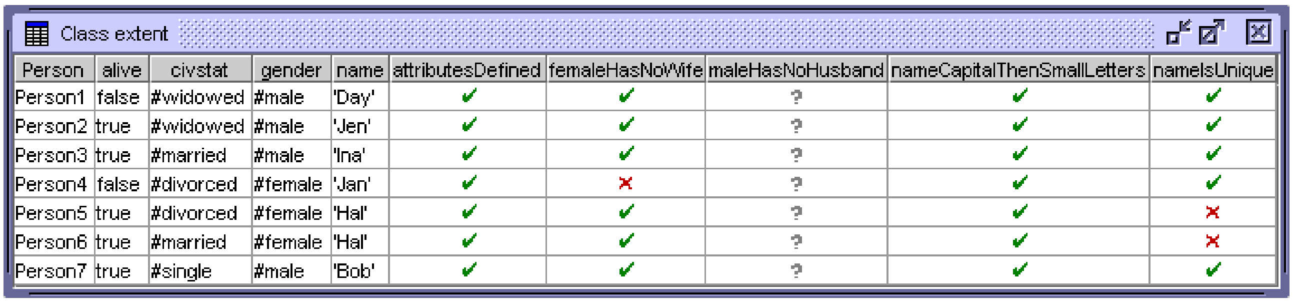
2 constraints failed. 100%

```
Person.allInstances->forAll(self : Person | ((self.gender = #female) implies self.wife.isUndefined)) = false
    Person.allInstances = Set{@Person1,@Person2,@Person3,@Person4,@Person5,@Person6,@Person7}
  ((self.gender = #female) implies self.wife.isUndefined) = true
  ((self.gender = #female) implies self.wife.isUndefined) = true
  ((self.gender = #female) implies self.wife.isUndefined) = true
  ((self.gender = #female) implies self.wife.isUndefined) = false
      (self.gender = #female) = true
          self.gender = #female
              self = @Person4
          #female = #female
      self.wife.isUndefined = false
          self.wife = @Person2
              self = @Person4
```

```
checking structure...
Multiplicity constraint violation in association `Marriage':
  Object `Person2' of class `Person' is connected to 2 objects of class `Person'
  but the multiplicity is specified as `0..1'.
checking structure, found errors.
```

++

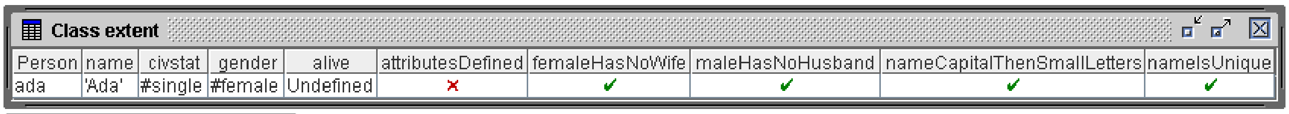
`crowd_classExtent.gif`


Class extent

Person	alive	civstat	gender	name	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
Person1	false	#widowed	#male	'Day'	✓	✓	?	✓	✓
Person2	true	#widowed	#male	'Jen'	✓	✓	?	✓	✓
Person3	true	#married	#male	'Ina'	✓	✓	?	✓	✓
Person4	false	#divorced	#female	'Jan'	✓	✗	?	✓	✓
Person5	true	#divorced	#female	'Hal'	✓	✓	?	✓	✗
Person6	true	#married	#female	'Hal'	✓	✓	?	✓	✗
Person7	true	#single	#male	'Bob'	✓	✓	?	✓	✓

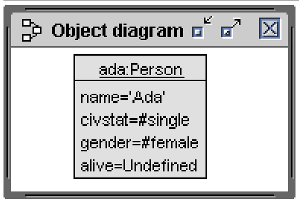
```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

E. Scenario independence

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

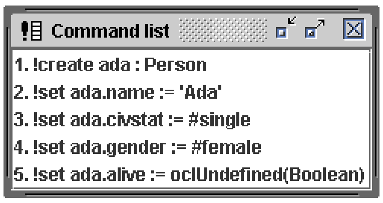
independence_attributesDefined.gif



Class extent

Person	name	civstat	gender	alive	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
ada	'Ada'	#single	#female	Undefined	✗	✓	✓	✓	✓



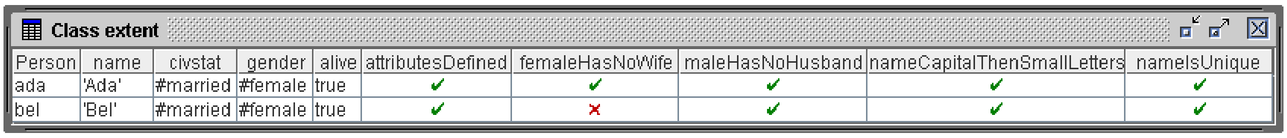


Command list

1. !create ada : Person
2. !set ada.name := 'Ada'
3. !set ada.civstat := #single
4. !set ada.gender := #female
5. !set ada.alive := oclUndefined(Boolean)

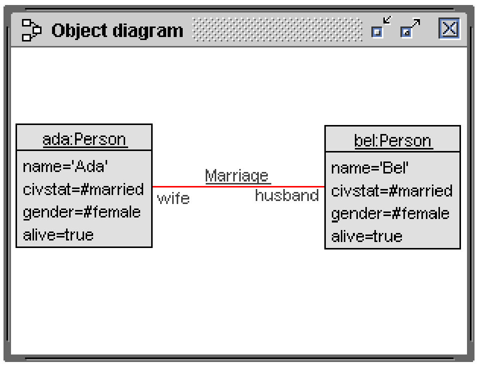
```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

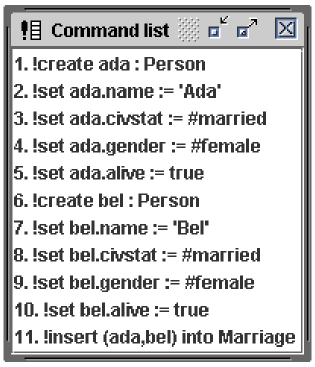
independence_femaleHasNoWife.gif



Class extent

Person	name	civstat	gender	alive	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
ada	'Ada'	#married	#female	true	✓	✓	✓	✓	✓
bel	'Bel'	#married	#female	true	✓	✗	✓	✓	✓



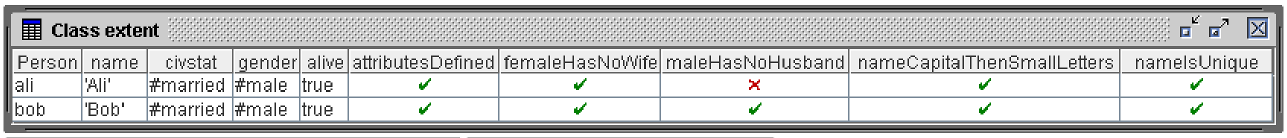


Command list

1. !create ada : Person
2. !set ada.name := 'Ada'
3. !set ada.civstat := #married
4. !set ada.gender := #female
5. !set ada.alive := true
6. !create bel : Person
7. !set bel.name := 'Bel'
8. !set bel.civstat := #married
9. !set bel.gender := #female
10. !set bel.alive := true
11. !insert (ada,bel) into Marriage

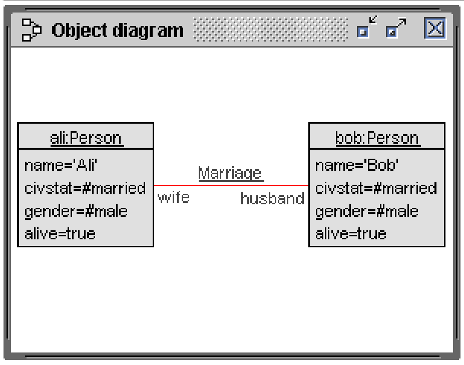
++

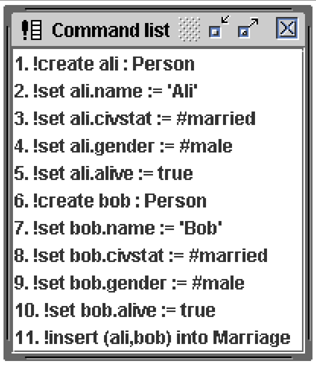
`independence_maleHasNoHusband.gif`



Class extent

Person	name	civstat	gender	alive	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
ali	'Ali'	#married	#male	true	✓	✓	✗	✓	✓
bob	'Bob'	#married	#male	true	✓	✓	✓	✓	✓



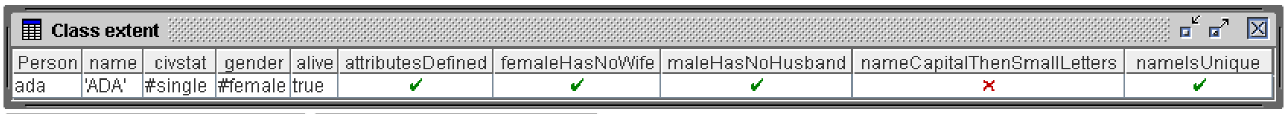


Command list

```
1. !create ali : Person
2. !set ali.name := 'Ali'
3. !set ali.civstat := #married
4. !set ali.gender := #male
5. !set ali.alive := true
6. !create bob : Person
7. !set bob.name := 'Bob'
8. !set bob.civstat := #married
9. !set bob.gender := #male
10. !set bob.alive := true
11. !insert (ali,bob) into Marriage
```

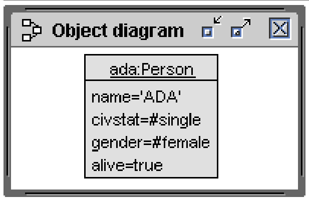
++

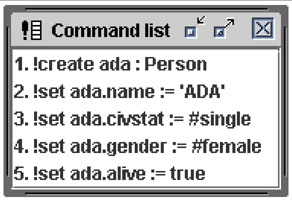
`independence_nameCapitalThenSmallLetters.gif`



Class extent

Person	name	civstat	gender	alive	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
ada	'ADA'	#single	#female	true	✓	✓	✓	✗	✓



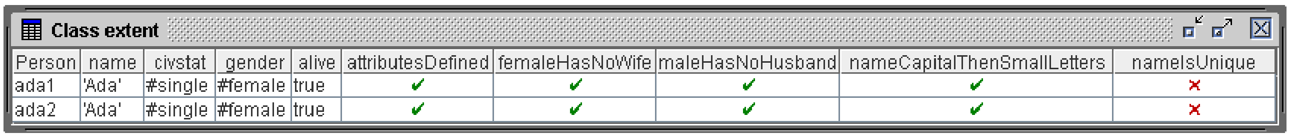


Command list

```
1. !create ada : Person
2. !set ada.name := 'ADA'
3. !set ada.civstat := #single
4. !set ada.gender := #female
5. !set ada.alive := true
```

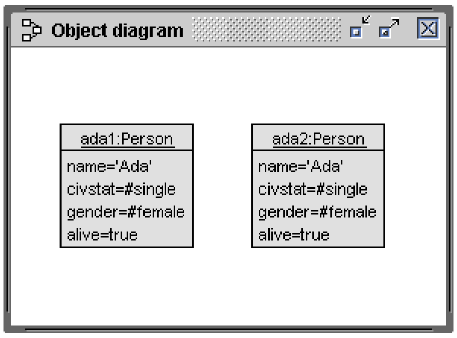
++

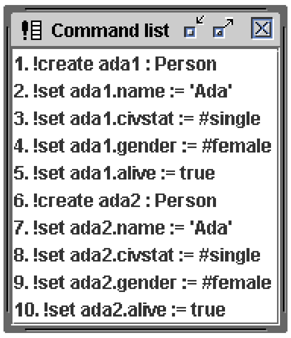
`independence_nameIsUnique.gif`



Class extent

Person	name	civstat	gender	alive	attributesDefined	femaleHasNoWife	maleHasNoHusband	nameCapitalThenSmallLetters	nameIsUnique
ada1	'Ada'	#single	#female	true	✓	✓	✓	✓	✗
ada2	'Ada'	#single	#female	true	✓	✓	✓	✓	✗





Command list

```
1. !create ada1 : Person
2. !set ada1.name := 'Ada'
3. !set ada1.civstat := #single
4. !set ada1.gender := #female
5. !set ada1.alive := true
6. !create ada2 : Person
7. !set ada2.name := 'Ada'
8. !set ada2.civstat := #single
9. !set ada2.gender := #female
10. !set ada2.alive := true
```

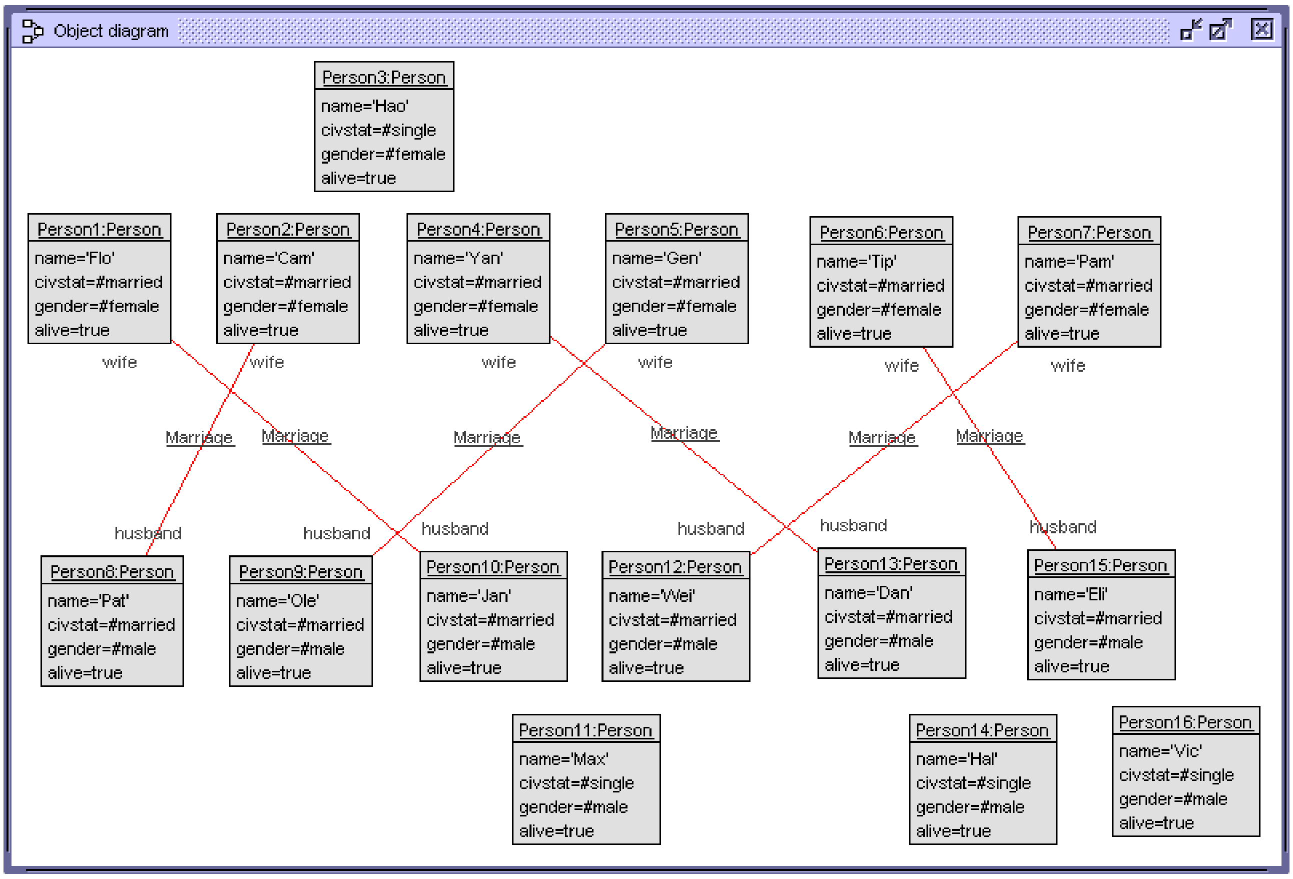
```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

F. Scenario world

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

world.pro

```
use> open civstat.use

use> gen start -r 2960 civstat.assl world(7,9,6)
use> gen result
     Random number generator was initialized with 2960.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3,Person4,Person5,Person6,
             Person7 : Person
     !create Person8,Person9,Person10,Person11,Person12,Person13,
             Person14,Person15,Person16 : Person
     !set Person1.name := 'Flo'
     !set Person1.civstat := #single
     !set Person1.gender := #female
     !set Person1.alive := true
     !set Person2.name := 'Cam'
     !set Person2.civstat := #single
     !set Person2.gender := #female
     !set Person2.alive := true
     !set Person3.name := 'Hao'
     !set Person3.civstat := #single
     !set Person3.gender := #female
     !set Person3.alive := true
     !set Person4.name := 'Yan'
     !set Person4.civstat := #single
     !set Person4.gender := #female
     !set Person4.alive := true
     !set Person5.name := 'Gen'
     !set Person5.civstat := #single
     !set Person5.gender := #female
     !set Person5.alive := true
     !set Person6.name := 'Tip'
     !set Person6.civstat := #single
     !set Person6.gender := #female
     !set Person6.alive := true
     !set Person7.name := 'Pam'
     !set Person7.civstat := #single
     !set Person7.gender := #female
     !set Person7.alive := true
     !set Person8.name := 'Pat'
     !set Person8.civstat := #single
     !set Person8.gender := #male
     !set Person8.alive := true
     !set Person9.name := 'Ole'
     !set Person9.civstat := #single
     !set Person9.gender := #male
     !set Person9.alive := true
     !set Person10.name := 'Jan'
     !set Person10.civstat := #single
     !set Person10.gender := #male
     !set Person10.alive := true
     !set Person11.name := 'Max'
     !set Person11.civstat := #single
     !set Person11.gender := #male
     !set Person11.alive := true
     !set Person12.name := 'Wei'
     !set Person12.civstat := #single
     !set Person12.gender := #male
     !set Person12.alive := true
     !set Person13.name := 'Dan'
     !set Person13.civstat := #single
     !set Person13.gender := #male
     !set Person13.alive := true
     !set Person14.name := 'Hal'
     !set Person14.civstat := #single
     !set Person14.gender := #male
     !set Person14.alive := true
     !set Person15.name := 'Eli'
     !set Person15.civstat := #single
     !set Person15.gender := #male
     !set Person15.alive := true
     !set Person16.name := 'Vic'
     !set Person16.civstat := #single
     !set Person16.gender := #male
     !set Person16.alive := true
     !set Person7.civstat := #married
     !set Person12.civstat := #married
     !insert (Person7,Person12) into Marriage
     !set Person6.civstat := #married
     !set Person15.civstat := #married
     !insert (Person6,Person15) into Marriage
     !set Person5.civstat := #married
     !set Person9.civstat := #married
     !insert (Person5,Person9) into Marriage
     !set Person4.civstat := #married
     !set Person13.civstat := #married
     !insert (Person4,Person13) into Marriage
     !set Person2.civstat := #married
     !set Person8.civstat := #married
     !insert (Person2,Person8) into Marriage
     !set Person1.civstat := #married
     !set Person10.civstat := #married
     !insert (Person1,Person10) into Marriage
use> gen result accept
     Generated result (system state) accepted.
```

```
use> check
     checking structure...
     checking invariants...
     checking invariant (1) `Person::attributesDefined': OK.
     checking invariant (2) `Person::femaleHasNoWife': OK.
     checking invariant (3) `Person::maleHasNoHusband': OK.
     checking invariant (4) `Person::nameCapitalThenSmallLetters': OK.
     checking invariant (5) `Person::nameIsUnique': OK.
     checked 5 invariants in 0.016s, 0 failures.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

world_objDia.gif



```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

world_query.pro

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

use> ?Person.allInstances
     Set{@Person1,@Person10,@Person11,@Person12,@Person13,@Person14,
         @Person15,@Person16,@Person2,@Person3,@Person4,@Person5,
         @Person6,@Person7,@Person8,@Person9} : Set(Person)

use> ?Person.allInstances.name
     Bag{'Cam','Dan','Eli','Flo','Gen','Hal','Hao','Jan','Max','Ole',
         'Pam','Pat','Tip','Vic','Wei','Yan'} : Bag(String)

use> ?Person.allInstances->collect(p|Tuple{cs:p.civstat,gd:p.gender})
     Bag{Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#female},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#married,gd=#male},
         Tuple{cs=#single,gd=#female},
         Tuple{cs=#single,gd=#male},
         Tuple{cs=#single,gd=#male},
         Tuple{cs=#single,gd=#male}} :
           Bag(Tuple(cs:CivilStatus,gd:Gender))

use> ?Person.allInstances->
use>   select(p|p.name.substring(1,1)<='M')->
use>     collectNested(p|
use>       Sequence{p.name,
use>                if p.gender=#female then 'F' else 'M' endif,
use>                p.civstat,
use>                p.spouse().name})
     Bag{Sequence{'Cam','F',#married,'Pat'},
         Sequence{'Dan','M',#married,'Yan'},
         Sequence{'Eli','M',#married,'Tip'},
         Sequence{'Flo','F',#married,'Jan'},
         Sequence{'Gen','F',#married,'Ole'},
         Sequence{'Hal','M',#single,Undefined},
         Sequence{'Hao','F',#single,Undefined},
         Sequence{'Jan','M',#married,'Flo'},
         Sequence{'Max','M',#single,Undefined}} : Bag(Sequence(OclAny))
```

```
use> ?Person.allInstances->
use>   select(p|p.name.substring(1,1)<='M')->
use>     collect(p|
use>       Sequence{p.name,
use>                if p.gender=#female then 'F' else 'M' endif,
use>                p.civstat,
use>                p.spouse().name})
     Bag{Undefined,Undefined,Undefined,#married,#married,#married,
         #married,#married,#married,#single,#single,#single,'Cam',
         'Dan','Eli','F','F','F','F','Flo','Flo','Gen','Hal','Hao',
         'Jan','Jan','M','M','M','M','M','Max','Ole','Pat','Tip',
         'Yan'} : Bag(OclAny)

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

use> ?Person.allInstances->forAll(p1,p2|p1<>p2 implies p1.name<>p2.name)
     true : Boolean

use> ?Person.allInstances->forAll(p1,p2|p1.name=p2.name implies p1=p2)
     true : Boolean

use> ?Person.allInstances->isUnique(name)
     true : Boolean

use> ?Person.allInstances->isUnique(gender)
     false : Boolean

use> ?Person.allInstances->one(p|p.name='Pam')
     true : Boolean

use> ?Person.allInstances->any(name='Pam').husband.name
     'Wei' : String

use> ?Person.allInstances.husband.name
     Bag{Undefined,Undefined,Undefined,Undefined,Undefined,Undefined,
         Undefined,Undefined,Undefined,Undefined,'Dan','Eli','Jan','Ole',
         'Pat','Wei'} : Bag(String)

use> ?Person.allInstances.husband.name->select(s|s.isUndefined())->size()
     10 : Integer

use> ?Person.allInstances->select(p|p.husband.isUndefined)->size()
     10 : Integer

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

use> ?Person.allInstances->sortedBy(p|p.name)
     Sequence{@Person2,@Person13,@Person15,@Person1,@Person5,@Person14,
              @Person3,@Person10,@Person11,@Person9,@Person7,@Person8,
              @Person6,@Person16,@Person12,@Person4} : Sequence(Person)
```

```
use> ?Person.allInstances->
use>   sortedBy(p|p.name)->
use>     iterate(p:Person;
use>       res:String=''|
use>       res.concat(if res='' then '' else ' ' endif).concat(p.name))
     'Cam Dan Eli Flo Gen Hal Hao Jan Max Ole Pam Pat Tip Vic Wei Yan'
     : String

use> ?Person.allInstances->
use>   iterate(p1,p2:Person;
use>           res:Boolean=true|
use>           res and (p1<>p2 implies p1.name<>p2.name))
     true : Boolean

use> ?Person.allInstances->
use>   iterate(p; res:Set(Person)=Set{}|
use>     if p.name.substring(1,1)<='M'
use>       then res->including(p) else res endif).name
     Bag{'Cam','Dan','Eli','Flo','Gen','Hal','Hao','Jan','Max'} :
     Bag(String)
```

+++

Exercise: One of the screenshots contains an OCL query where the stated query will not yield the stated result. Which screenshot is this?

+++