# USE 4 EIS

Martin Gogolla

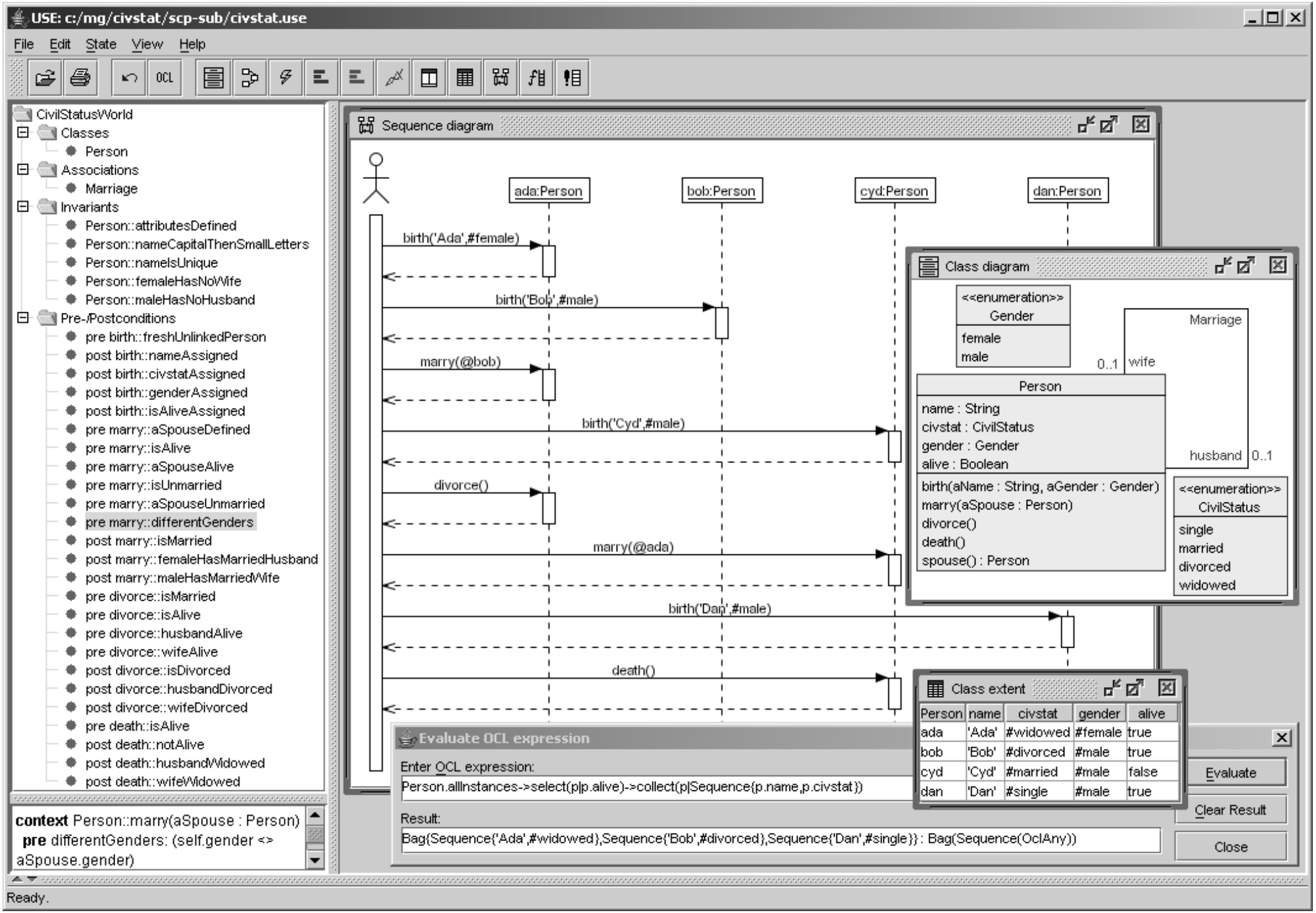
University of Bremen

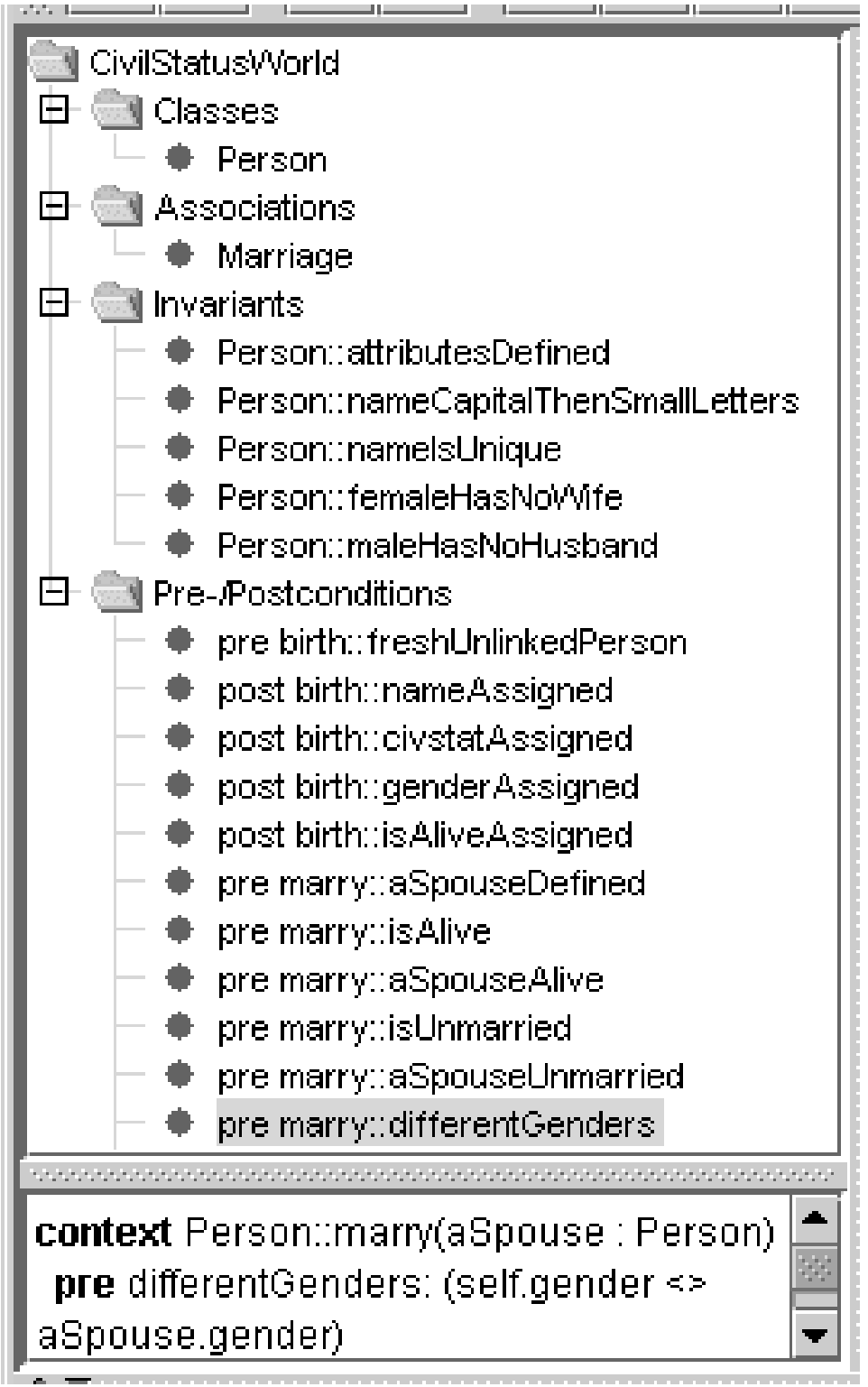
- OCL concepts
  - Basis: (graphical) UML concepts, in particular UML CD
  - Descriptive Language for expressions (logical values, objects, object collections)
  - Objects
  - **Navigation**
  - (Finite!) Collections: Set, Bag, Sequence, [OrderedSet]
  - Collection operations: forAll, exists, select, ...
  - Formal semantics: naive set-theoretic (PhD thesis Mark Richters [HB], also part of the OMG standard)
- OCL applications
  - UML metamodel (syntax of the UML)
  - Other (meta-)models (CWM, MOF, ODM, ER/RE), ...

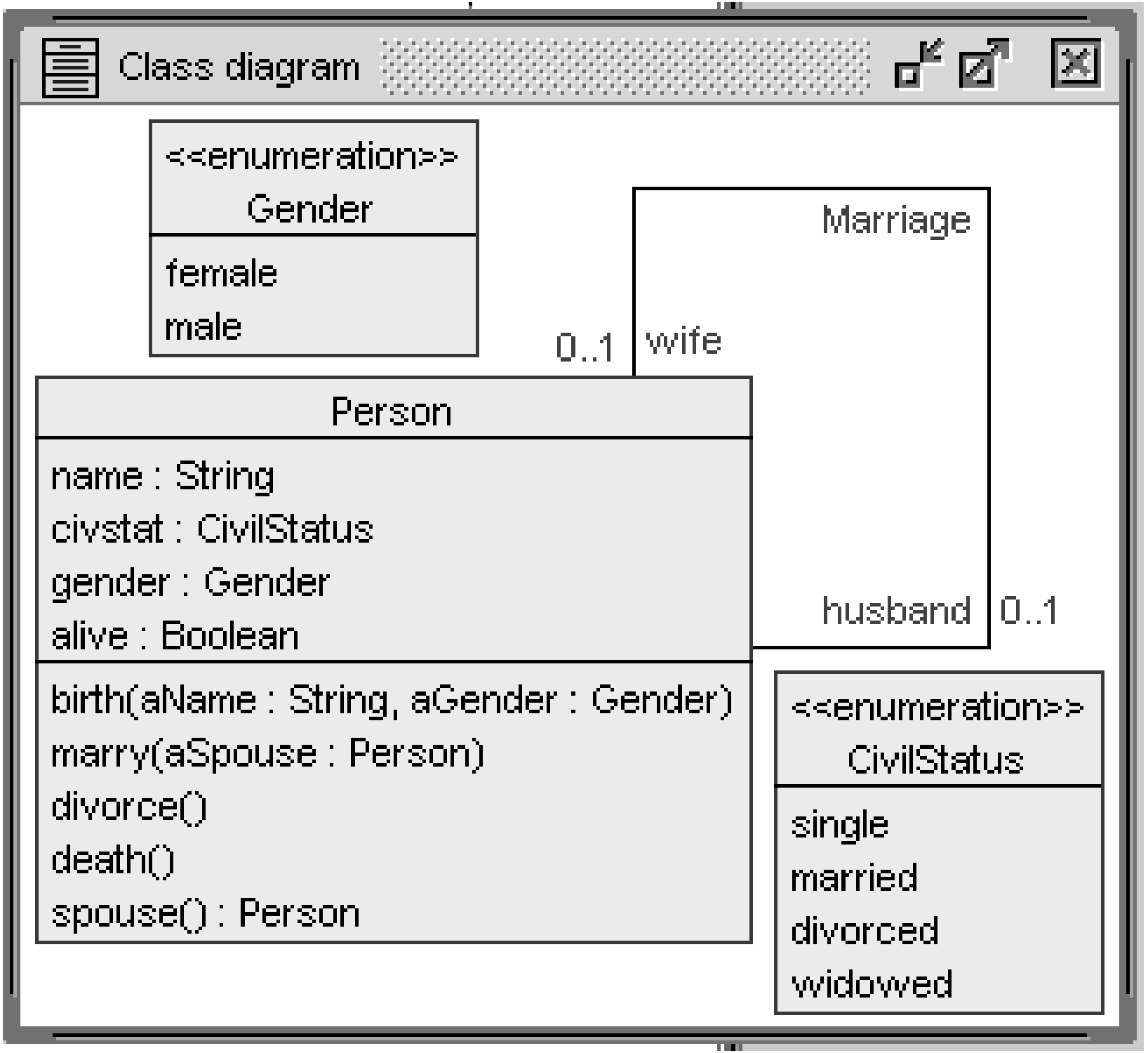
- UML tools with OCL support
  - Poseidon, ArgoUML, ...
  - MagicDraw, MaxUML, Together, XMF-Mosaic, ...
- OCL tools [QVT tools with OCL emerging]
  - Dresden OCL Compiler (OCL 2 Java)
  - Octopus (Warmer/Kleppe; syntax check & code generator)
  - KeY (Karlsruhe; OCL prover embedded in Together); OCL & verification also done in Zürich (Isabelle) and Kiel (PVS?)
  - OCLE (Romania; validation tool)
  - USE: UML Specification Environment (validation and 'certification' tool)
  - ...

- USE allows to
  - Get confidence in models (formal descriptions) by 'testing' it with scenarios
  - Check consistency of models (invariants); by constructing an Object Diagram
  - Show independency of invs (no invariant follows from the others); by constructing a state violating an invariant INV but satisfying all other invariants (a state satisfying 'negated INV' and ...)
  - Check whether property PROP follows from invariants; by showing it is not possible to construct a state where all invariants hold and the negation of PROP holds; drawback of the technique: search space for state construction has to be restricted by (expressed as) an ASSL program (A Snapshot Sequence Language)

- USE supports class diagrams
- invariants, pre/posts of operations, op definitions
  - Construct object diagram (system state) explicitly (create & destroy objects & links, set attributes)
  - Check invariants; inspect details with the 'Evaluation Browser', 'Object diagram', 'OCL Evaluation', ...
  - Construct operation call sequence
  - Check pre/postconditions & invariants
  - Generate object diagram descriptively (give desired properties of the object diagram)
  - Describe the search space (a set of object diagrams) by enumerating it with an 'ASSL' program


USE: c:/mg/civstat/scp-sub/civstat.use
File Edit State View Help
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)
Sequence diagram
ada:Person
bob:Person
cyd:Person
dan:Person
birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()
Class diagram
<<enumeration>>
Gender
female
male
Marriage
0..1
wife
husband
0..1
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
<<enumeration>>
CivilStatus
single
married
divorced
widowed
Class extent
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
Evaluate OCL expression
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))
Evaluate
Clear Result
Close
Ready.


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


Class diagram
<<enumeration>>
Gender
female
male
Marriage
0..1
wife
husband
0..1
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
<<enumeration>>
CivilStatus
single
married
divorced
widowed

```
model CivilStatusWorld

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

class Person

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

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

```
birth(aName:String, aGender:Gender)
marry(aSpouse:Person)
divorce()
death()

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

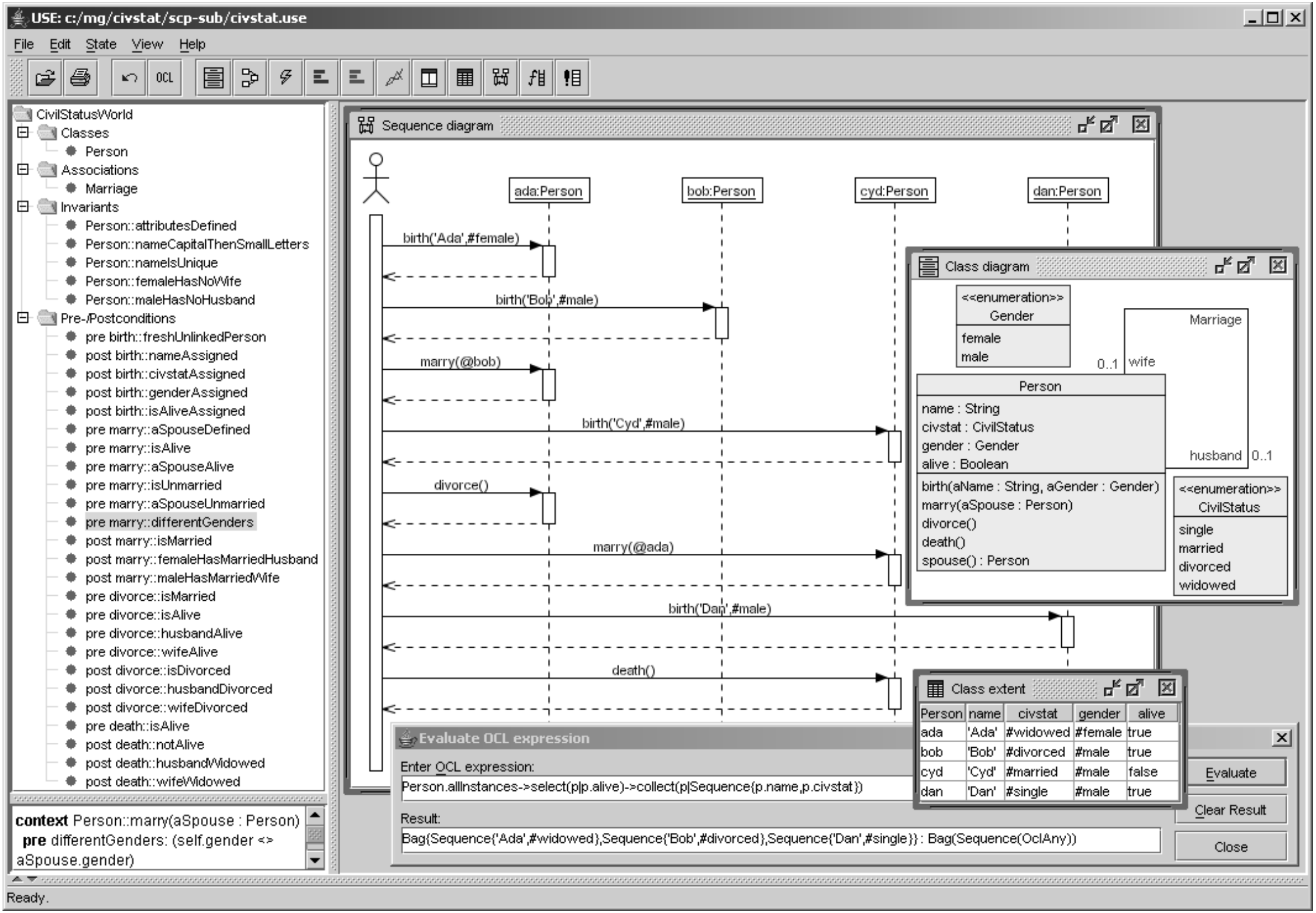
```
marry(aSpouse:Person)
pre  aSpouseDefined: aSpouse.isDefined
pre  isAlive: alive
pre  aSpouseAlive: aSpouse.alive
pre  isUnmarried: civstat<>#married
pre  aSpouseUnmarried: aSpouse.civstat<>#married
pre  differentGenders: gender<>aSpouse.gender
post isMarried: civstat=#married
post femaleHasMarriedHusband: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

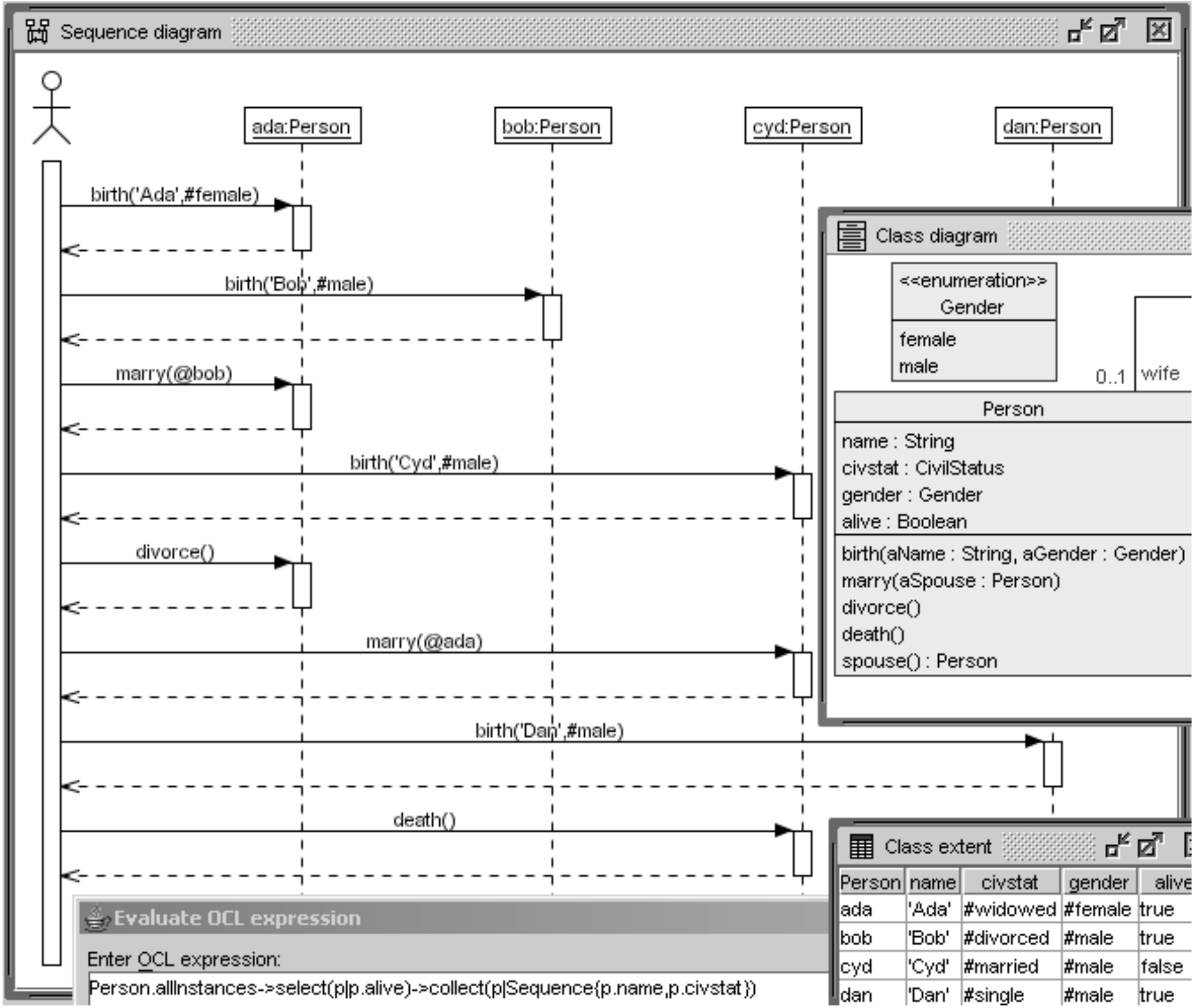
```
constraints
  inv attributesDefined: name.isDefined and
      civstat.isDefined and
      gender.isDefined and alive.isDefined

  inv nameCapitalThenSmallLetters:
      let small:Set(String)=
        Set{'a','b','c', ... ,'x','y','z'} in
      let capital:Set(String)=
        Set{'A','B','C', ... ,'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
```

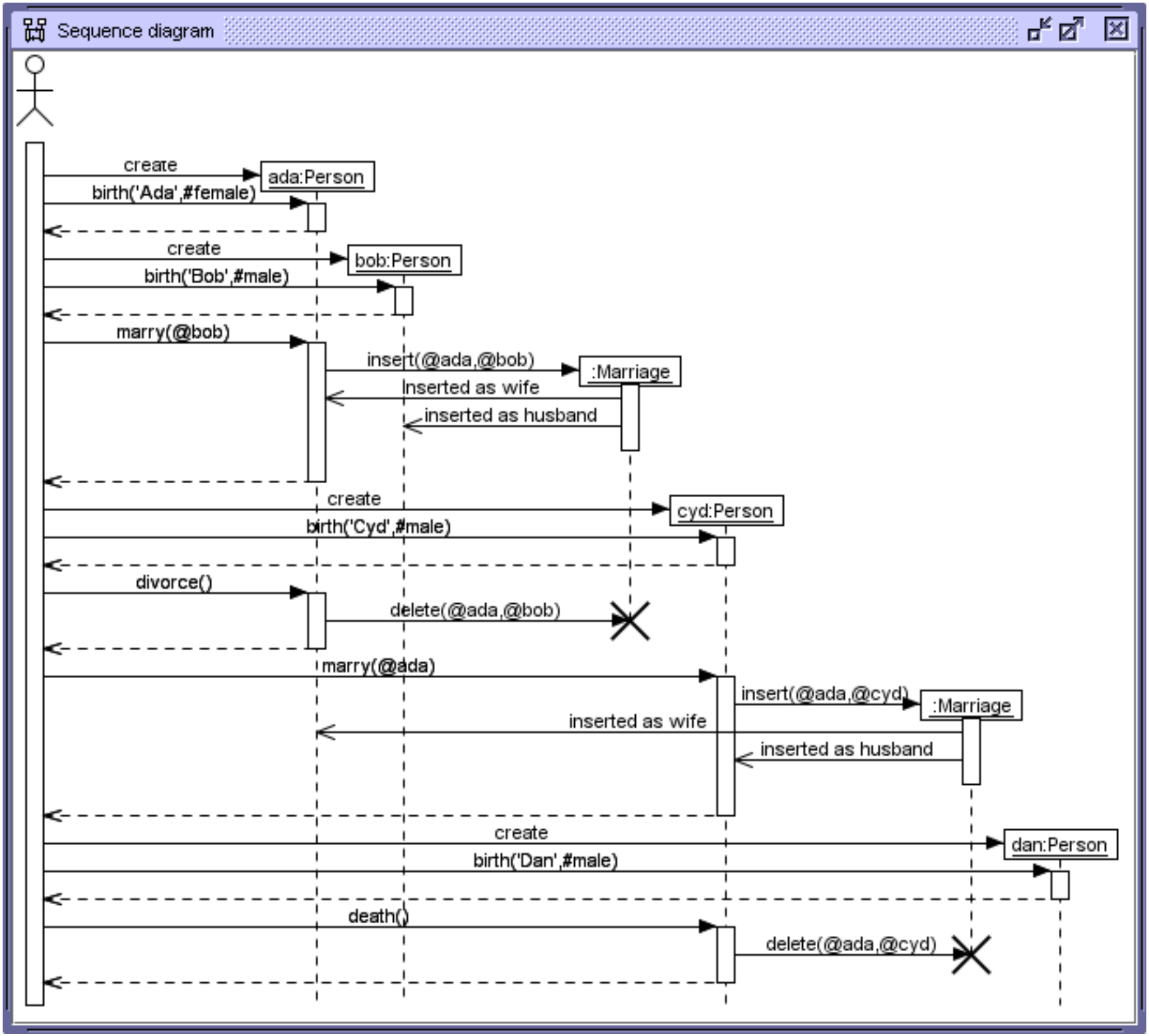
```
model NameWorld

class Person01
attributes
  name:String
constraints
  inv nameIsUnique01: Person01.allInstances->forAll(self2|
      self<>self2 implies self.name<>self2.name)
end

class Person02
attributes
  name:String
constraints
  inv nameIsUnique02: Person02.allInstances->forAll(self2:Person02|
      self<>self2 implies self.name<>self2.name)
end

class Person03
attributes
  name:String
constraints
  inv nameIsUnique03: Person03.allInstances->forAll(self2|
      not(self.name<>self2.name) implies not(self<>self2))
end

class Person04
attributes
  name:String
constraints
  inv nameIsUnique04: Person04.allInstances->forAll(self2|
      self.name=self2.name implies self=self2)
end
```

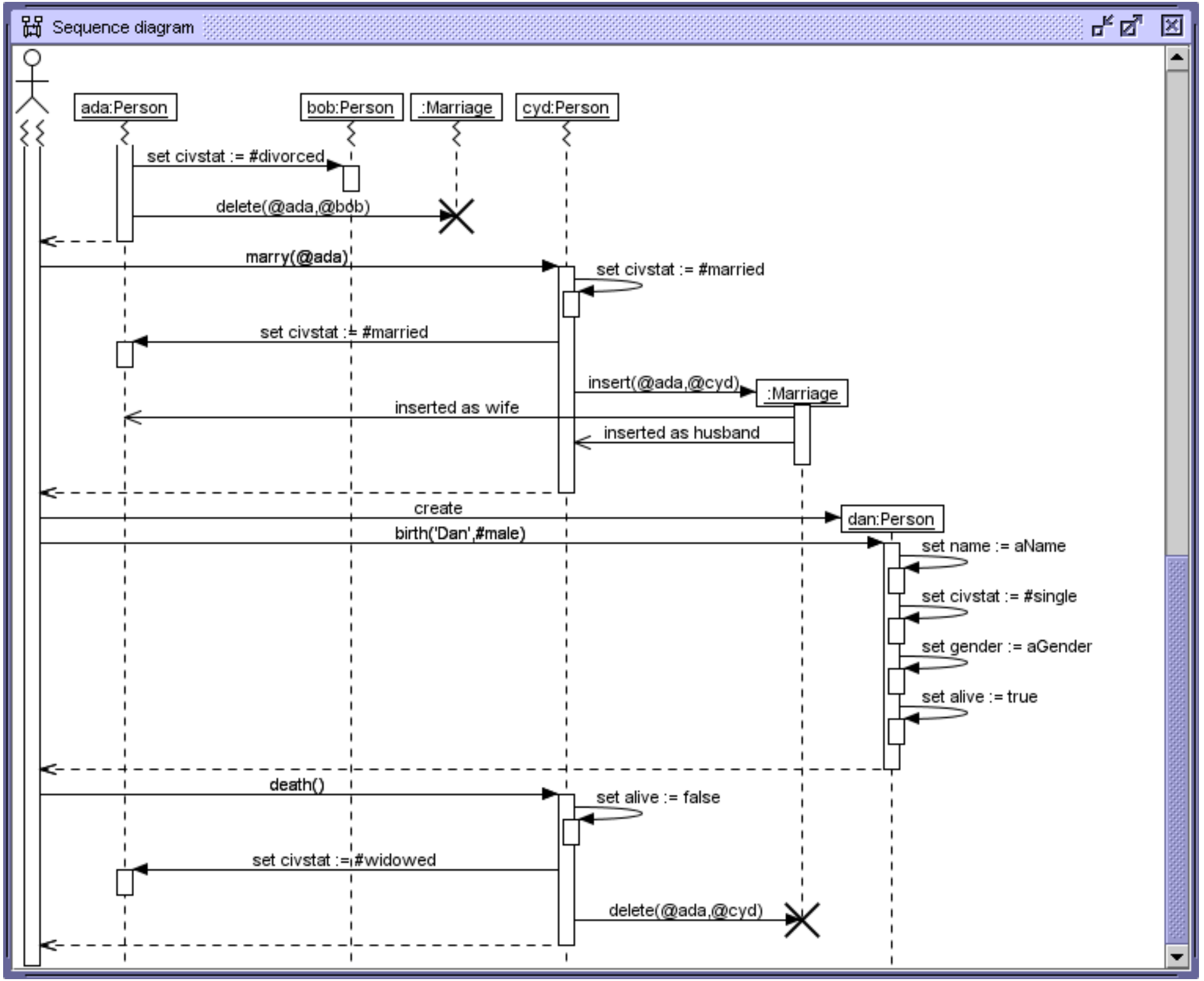
```
class Person
attributes
  name:String
end

constraints

context Person inv nameIsUnique05:
  Person.allInstances->forAll(self2|
    self<>self2 implies self.name<>self2.name)

context self:Person inv nameIsUnique06:
  Person.allInstances->forAll(self2|
    self<>self2 implies self.name<>self2.name)

-- Person.allInstances->forAll(self|
--   Person.allInstances->forAll(self2|
--     self<>self2 implies self.name<>self2.name)
--   )

context Person inv nameIsUnique07:
  Person.allInstances->forAll(p1,p2|
    p1<>p2 implies p1.name<>p2.name)

-- Person.allInstances->forAll(self|
--   Person.allInstances->forAll(p1,p2|
--     p1<>p2 implies p1.name<>p2.name)
--   )
```

```
context Person inv nameAda:
  Person.allInstances->exists(p|p.name='Ada')

-- Person.allInstances->forAll(self|
--   Person.allInstances->exists(p|p.name='Ada')
--   )

-- not(not(nameIsUnique06))
context self:Person inv nameIsUnique08:
  not(not(Person.allInstances->forAll(self2|
    self<>self2 implies self.name<>self2.name)))

context self:Person inv nameIsUnique09:
  not(Person.allInstances->exists(self2|
    not(self<>self2 implies self.name<>self2.name)))

context self:Person inv nameIsUnique10:
  not(Person.allInstances->exists(self2|
    self<>self2 and self.name=self2.name))

context self:Person inv nameIsUnique11:
  Person.allInstances->isUnique(p|p.name)

-- Person.allInstances->forAll(self|
--   Person.allInstances->isUnique(p|p.name)
--   )

context Person inv nameIsUnique12:
  Person.allInstances->isUnique(name)

-- there are further possibilities ...
```

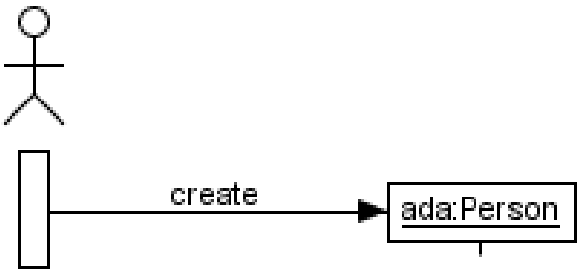

USE: c:/mg/civstat/scp-sub/civstat.use
File Edit State View Help
CivilStatusWorld
Classes
Person
Associations
Marriage
Invariants
Person::attributesDefined
Person::nameCapitalThenSmallLetters
Person::nameIsUnique
Person::femaleHasNoWife
Person::maleHasNoHusband
Pre-/Postconditions
pre birth::freshUnlinkedPerson
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)
Sequence diagram
ada:Person
bob:Person
cyd:Person
dan:Person
birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()
Class diagram
<<enumeration>>
Gender
female
male
Marriage
0..1
wife
husband
0..1
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
<<enumeration>>
CivilStatus
single
married
divorced
widowed
Class extent
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
Evaluate OCL expression
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))
Evaluate
Clear Result
Close
Ready.


Sequence diagram
ada:Person
bob:Person
cyd:Person
dan:Person
birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()
Class diagram
<<enumeration>>
Gender
female
male
0..1
wife
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
Class extent
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
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(p|p.alive)->collect(p|Sequence{p.name,p.civstat})

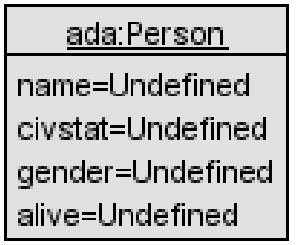

Sequence diagram
create
ada:Person
birth('Ada',#female)
create
bob:Person
birth('Bob',#male)
marry(@bob)
insert(@ada,@bob)
:Marriage
inserted as wife
inserted as husband
create
cyd:Person
birth('Cyd',#male)
divorce()
delete(@ada,@bob)
marry(@ada)
insert(@ada,@cyd)
:Marriage
inserted as wife
inserted as husband
create
dan:Person
birth('Dan',#male)
death()
delete(@ada,@cyd)


Sequence diagram
ada:Person
bob:Person
:Marriage
cyd:Person
set civstat := #divorced
delete(@ada,@bob)
marry(@ada)
set civstat := #married
set civstat := #married
insert(@ada,@cyd)
:Marriage
inserted as wife
inserted as husband
create
dan:Person
birth('Dan',#male)
set name := aName
set civstat := #single
set gender := aGender
set alive := true
death()
set alive := false
set civstat := #widowed
delete(@ada,@cyd)

Sequence diagram



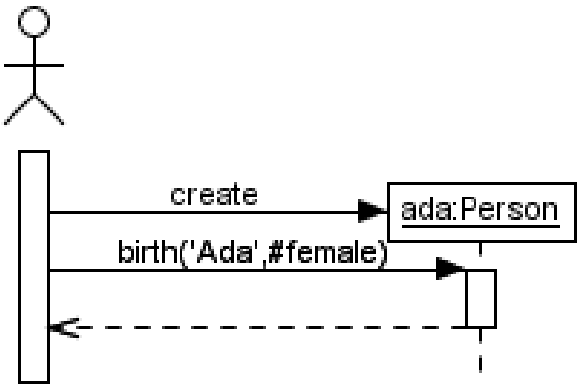

Object diagram



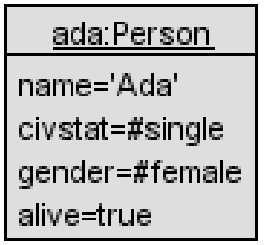

Class extent

| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | Undefined | Undefined | Undefined | Undefined |

Sequence diagram



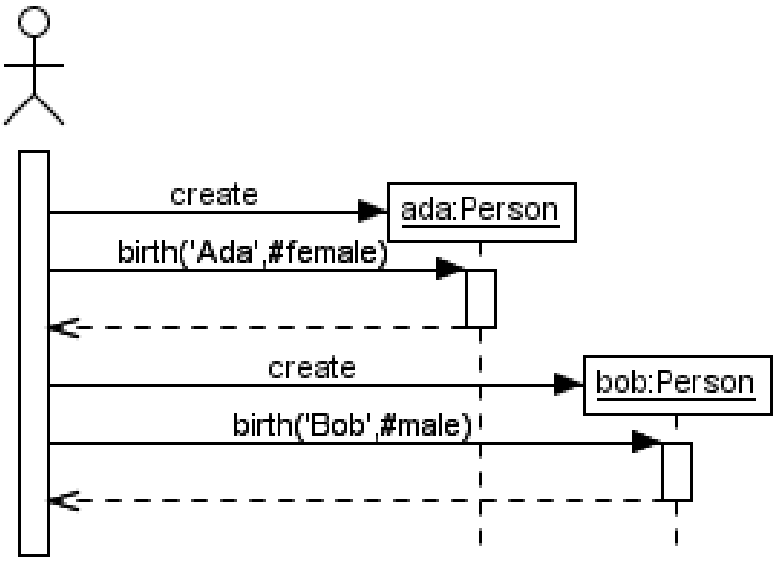

Object diagram



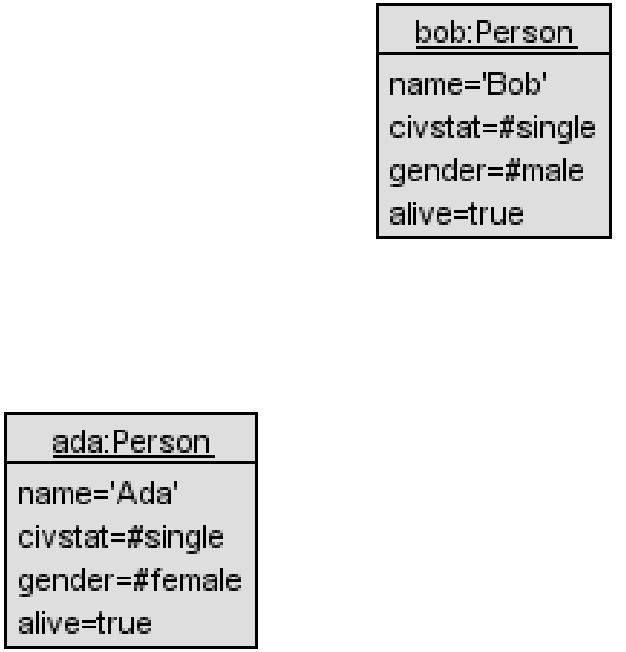

Class extent

| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #single | #female | true |

Sequence diagram



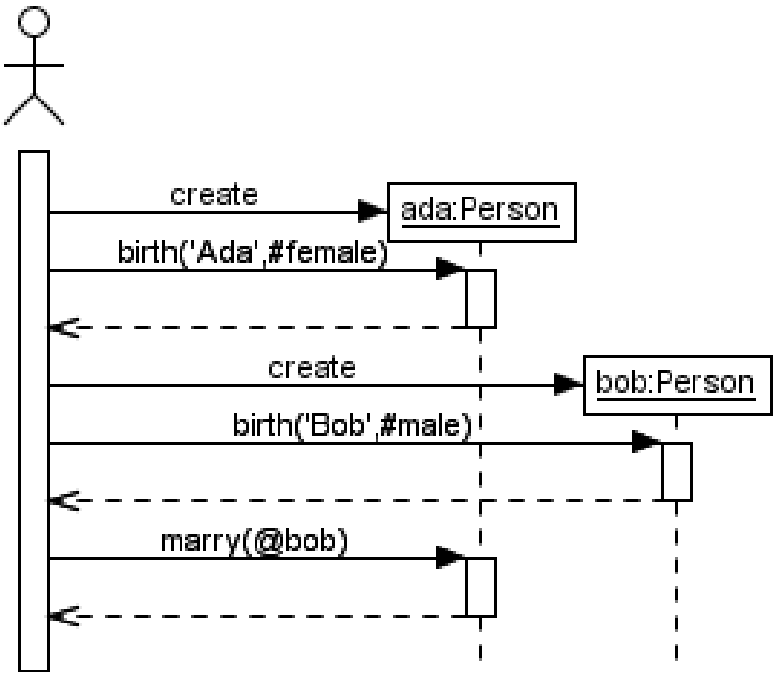

Object diagram



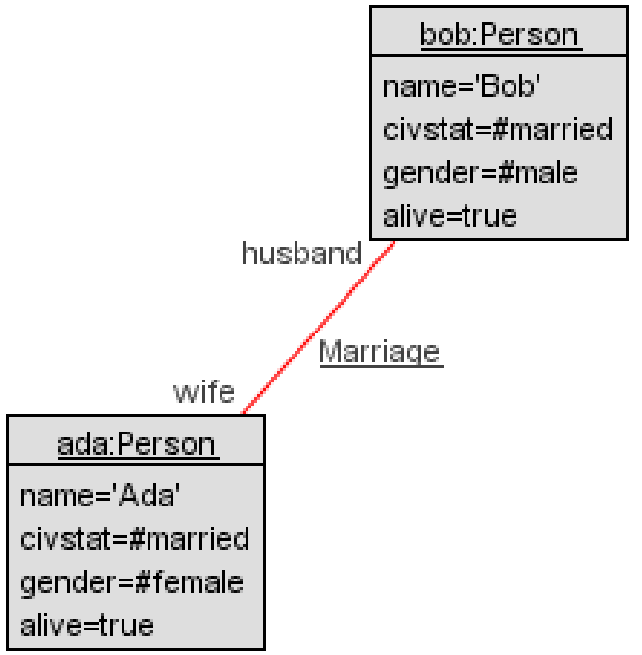

Class extent

| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #single | #female | true |
| bob | 'Bob' | #single | #male | true |

Sequence diagram



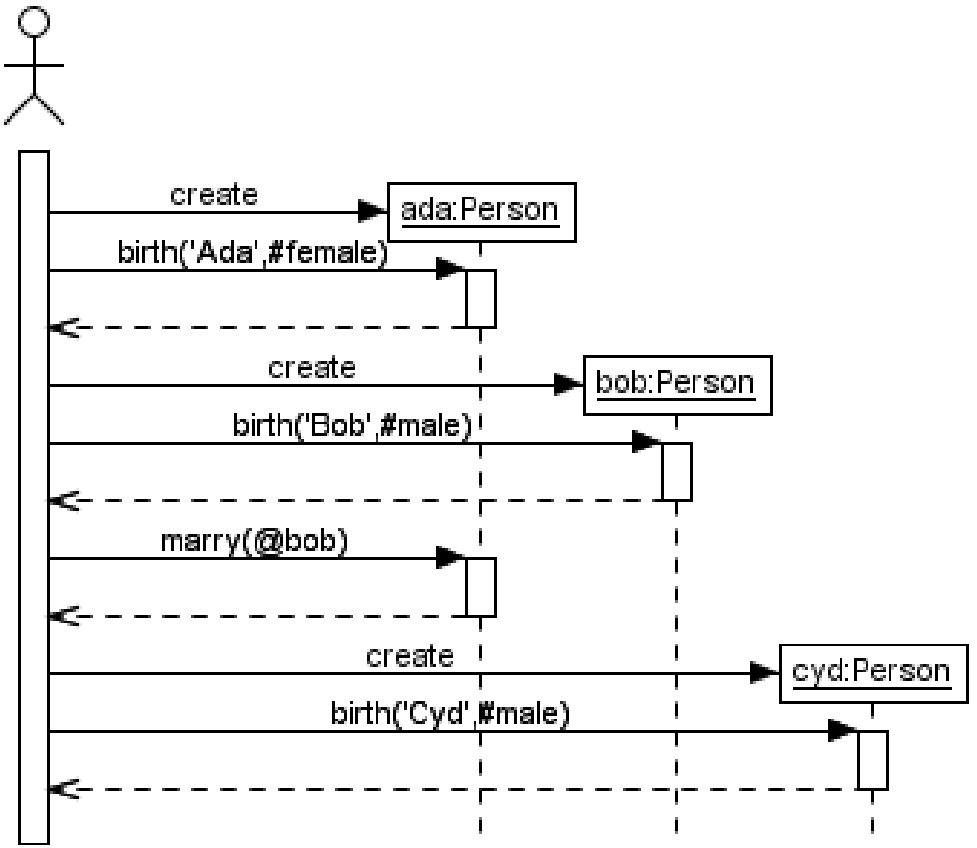

Object diagram



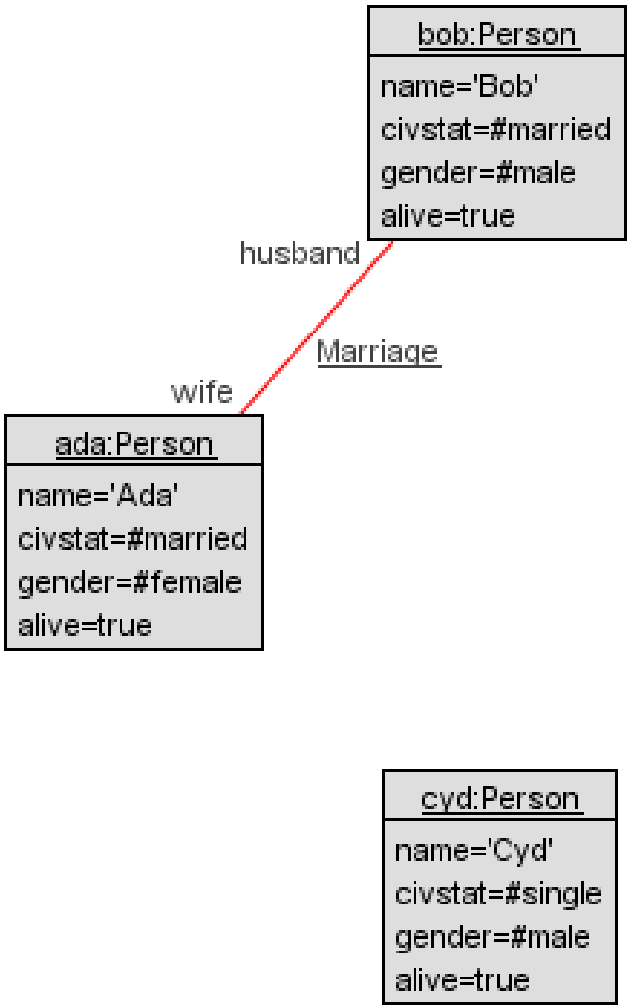

Class extent

| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #married | #female | true |
| bob | 'Bob' | #married | #male | true |

Sequence diagram

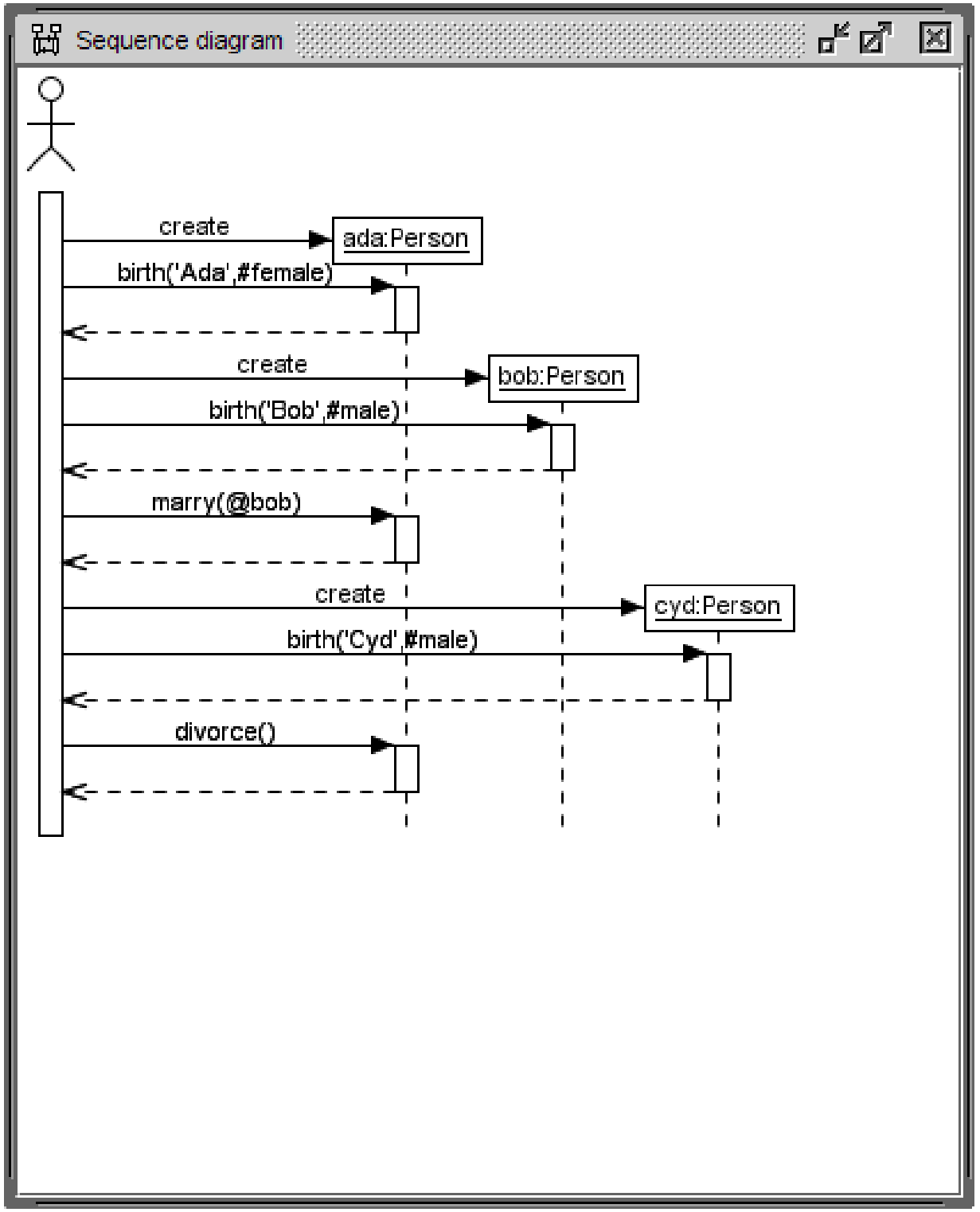


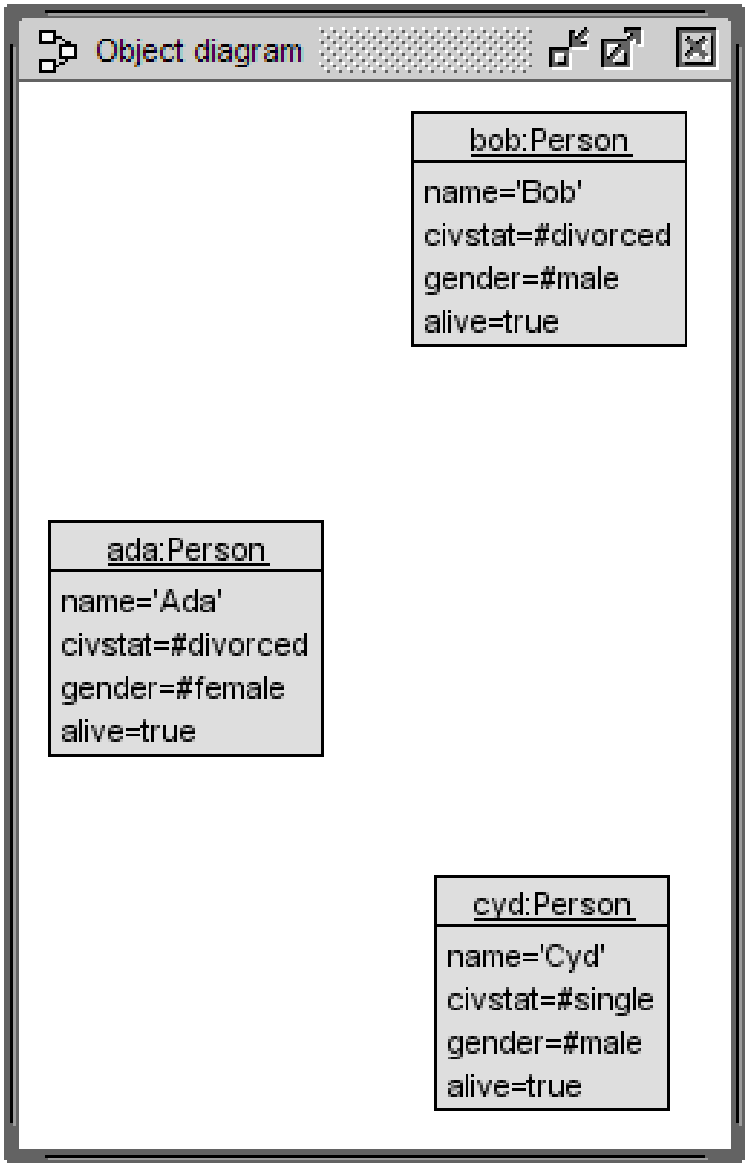

Object diagram




Class extent

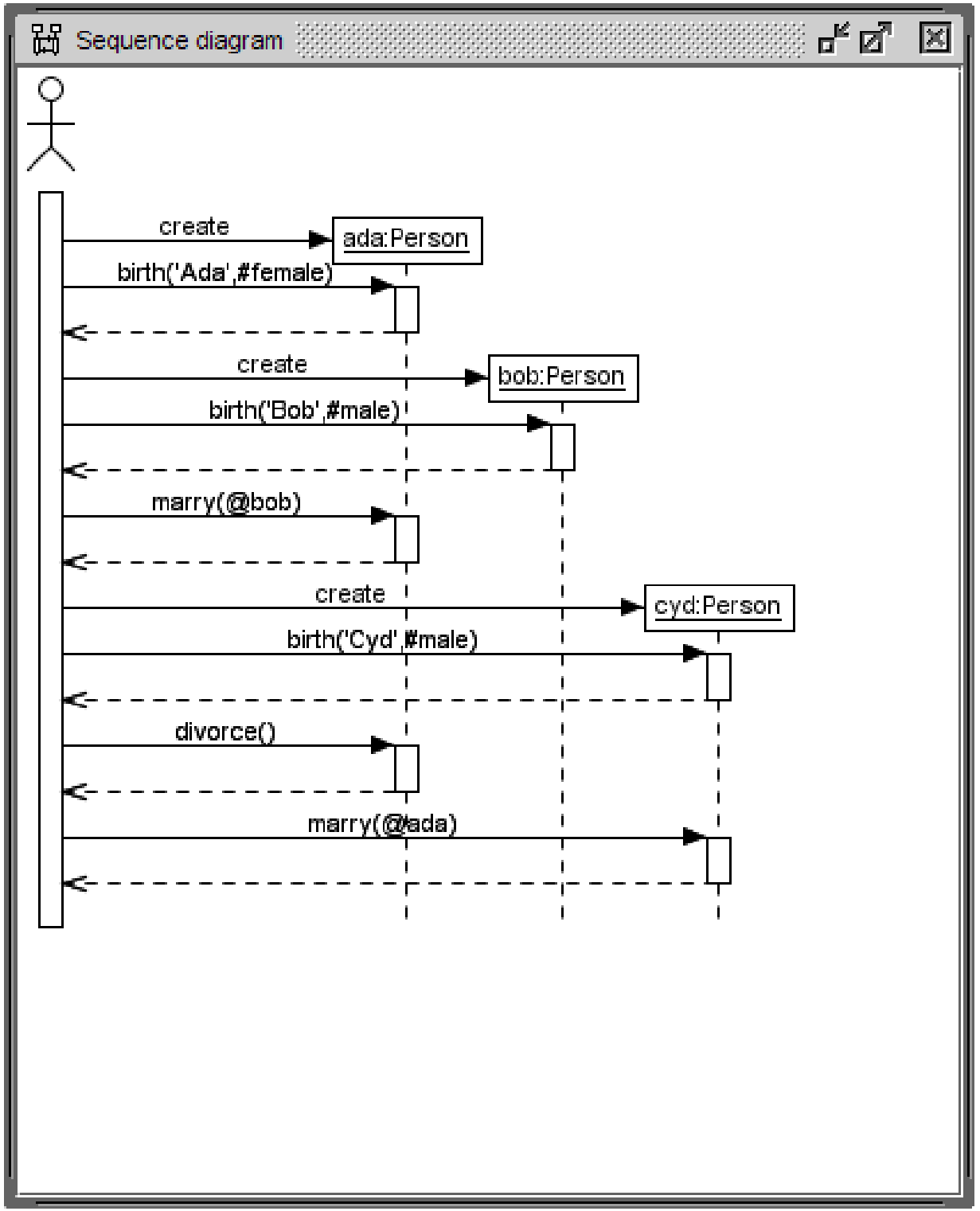
| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #married | #female | true |
| bob | 'Bob' | #married | #male | true |
| cyd | 'Cyd' | #single | #male | true |

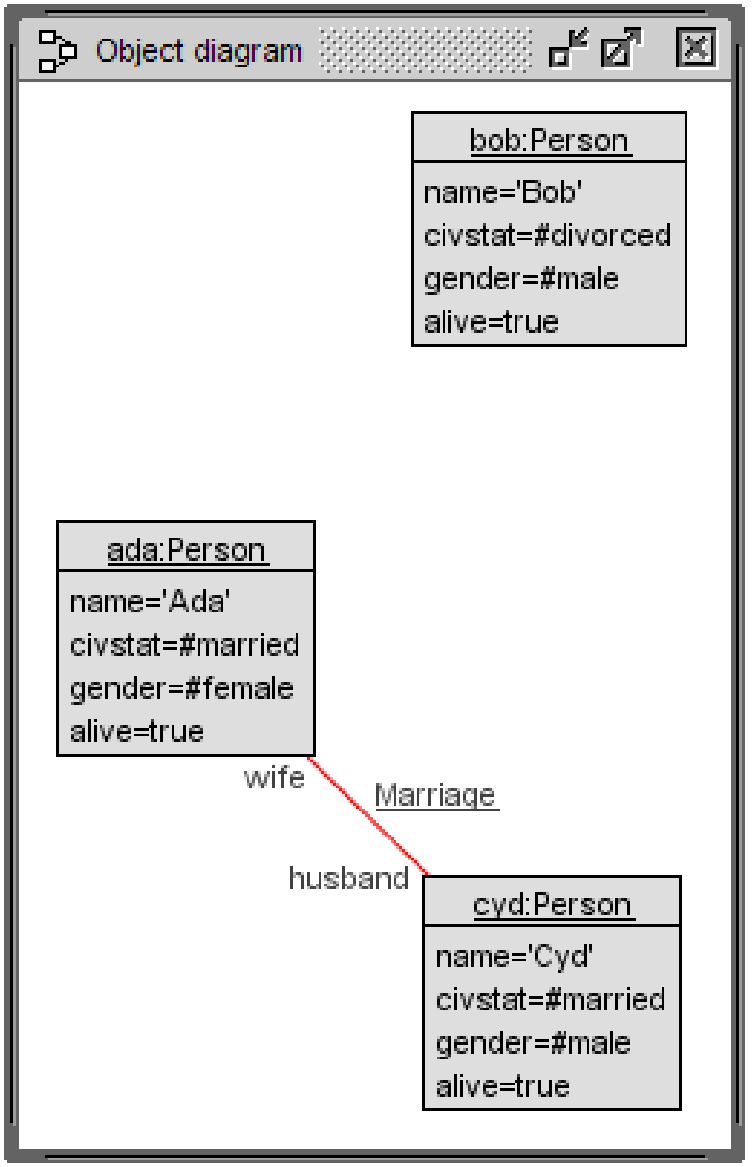



Class extent

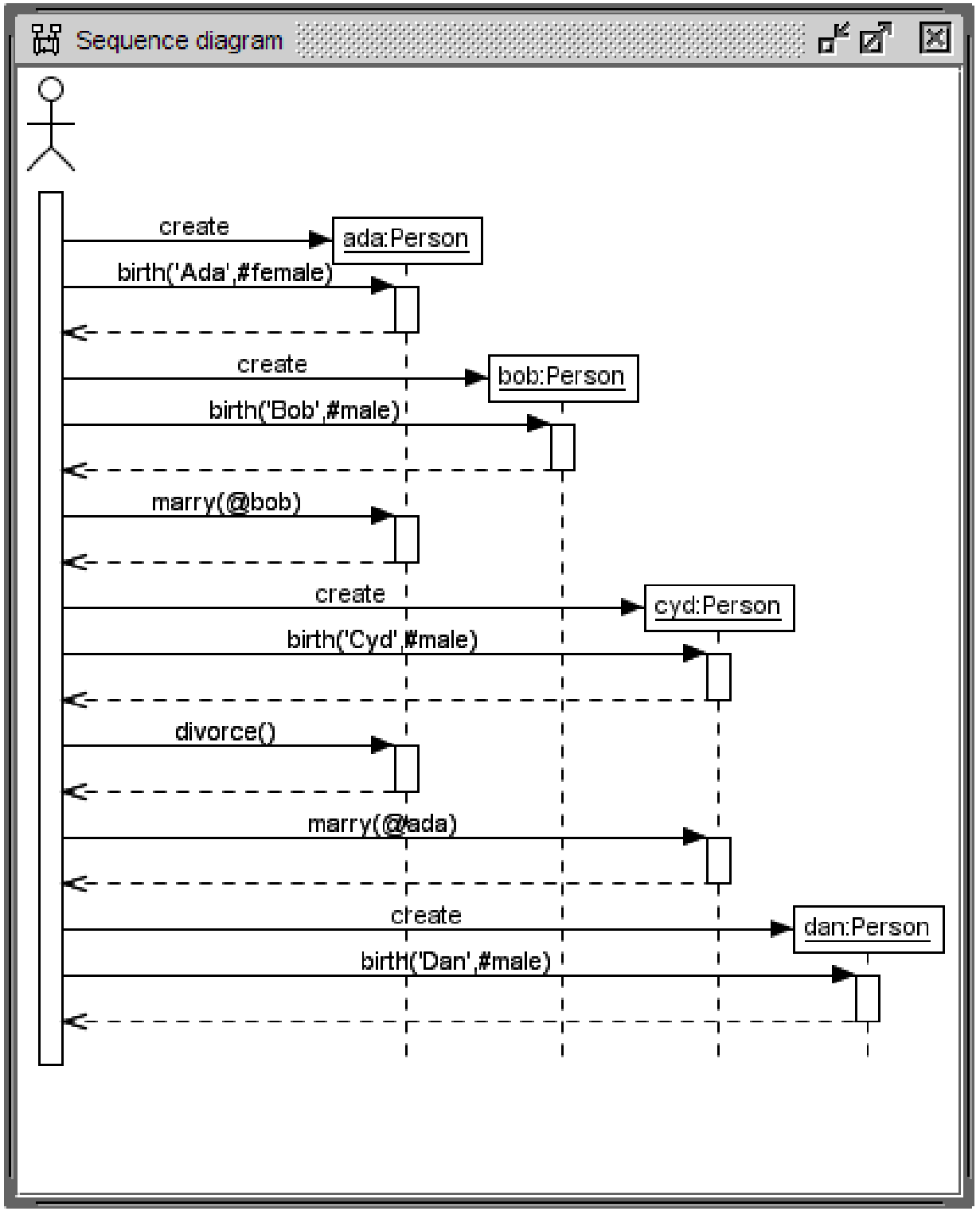
| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #divorced | #female | true |
| bob | 'Bob' | #divorced | #male | true |
| cyd | 'Cyd' | #single | #male | true |

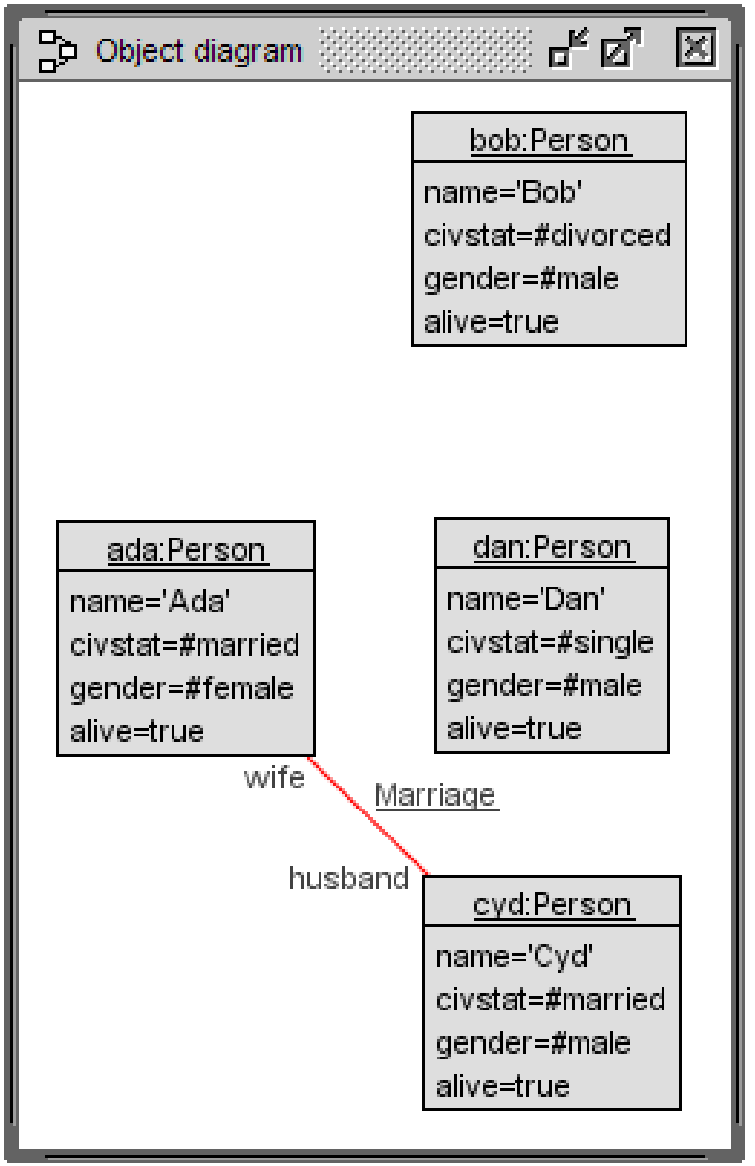



Class extent

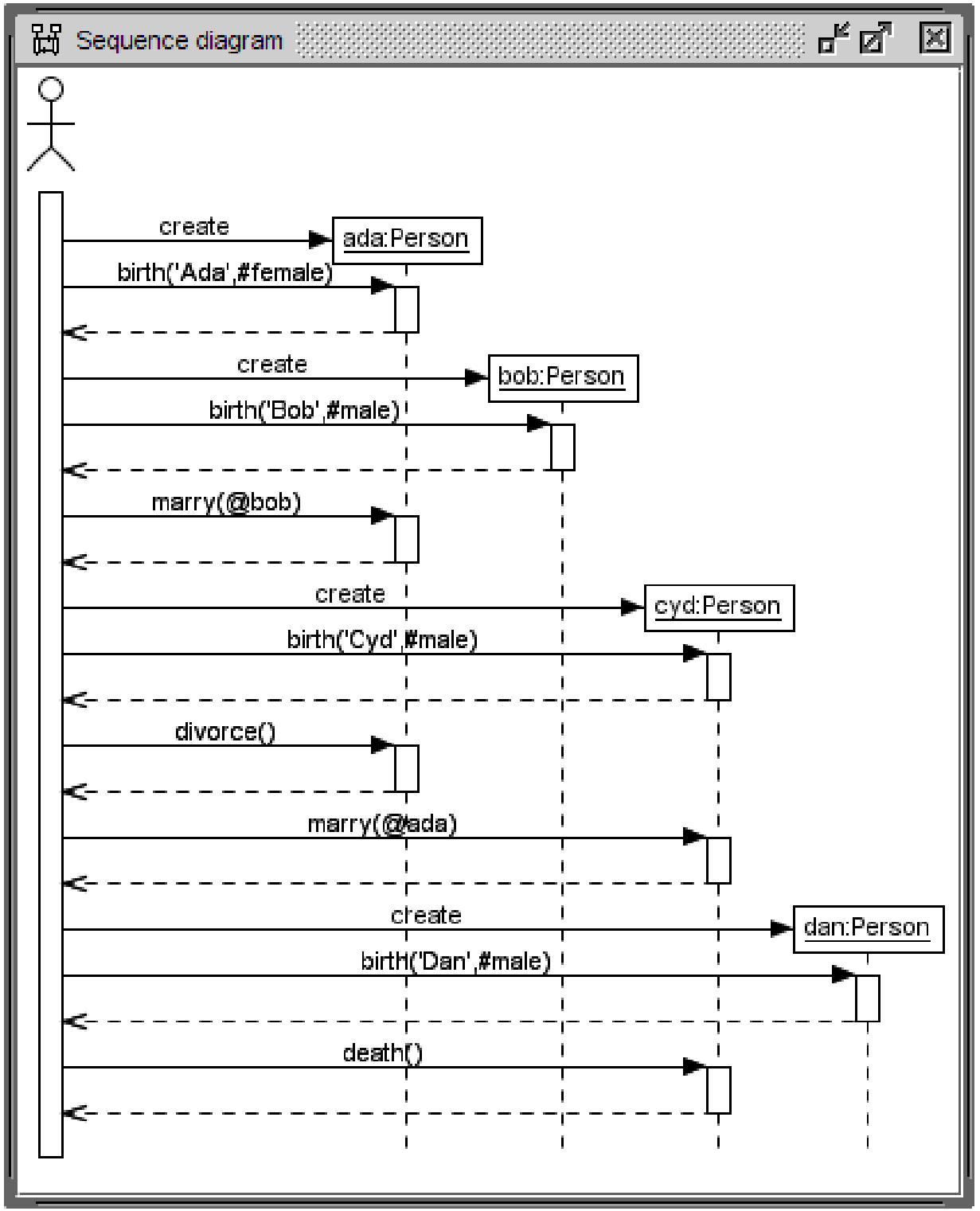
| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #married | #female | true |
| bob | 'Bob' | #divorced | #male | true |
| cyd | 'Cyd' | #married | #male | true |

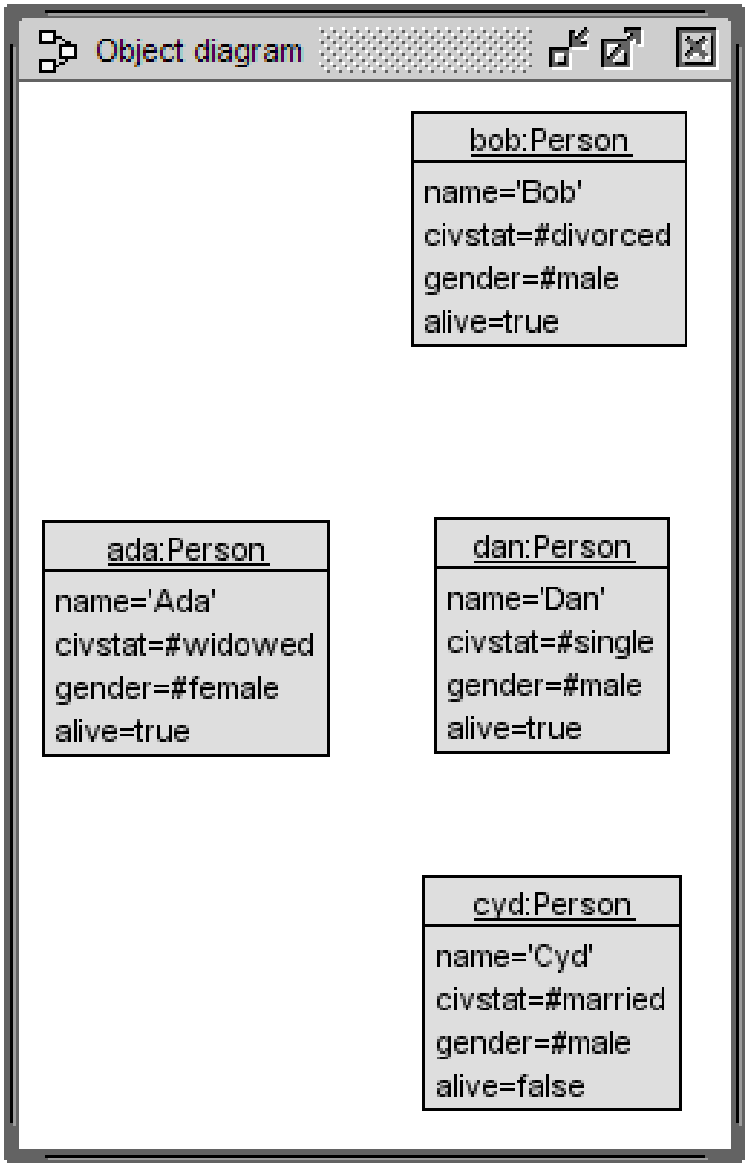



Class extent

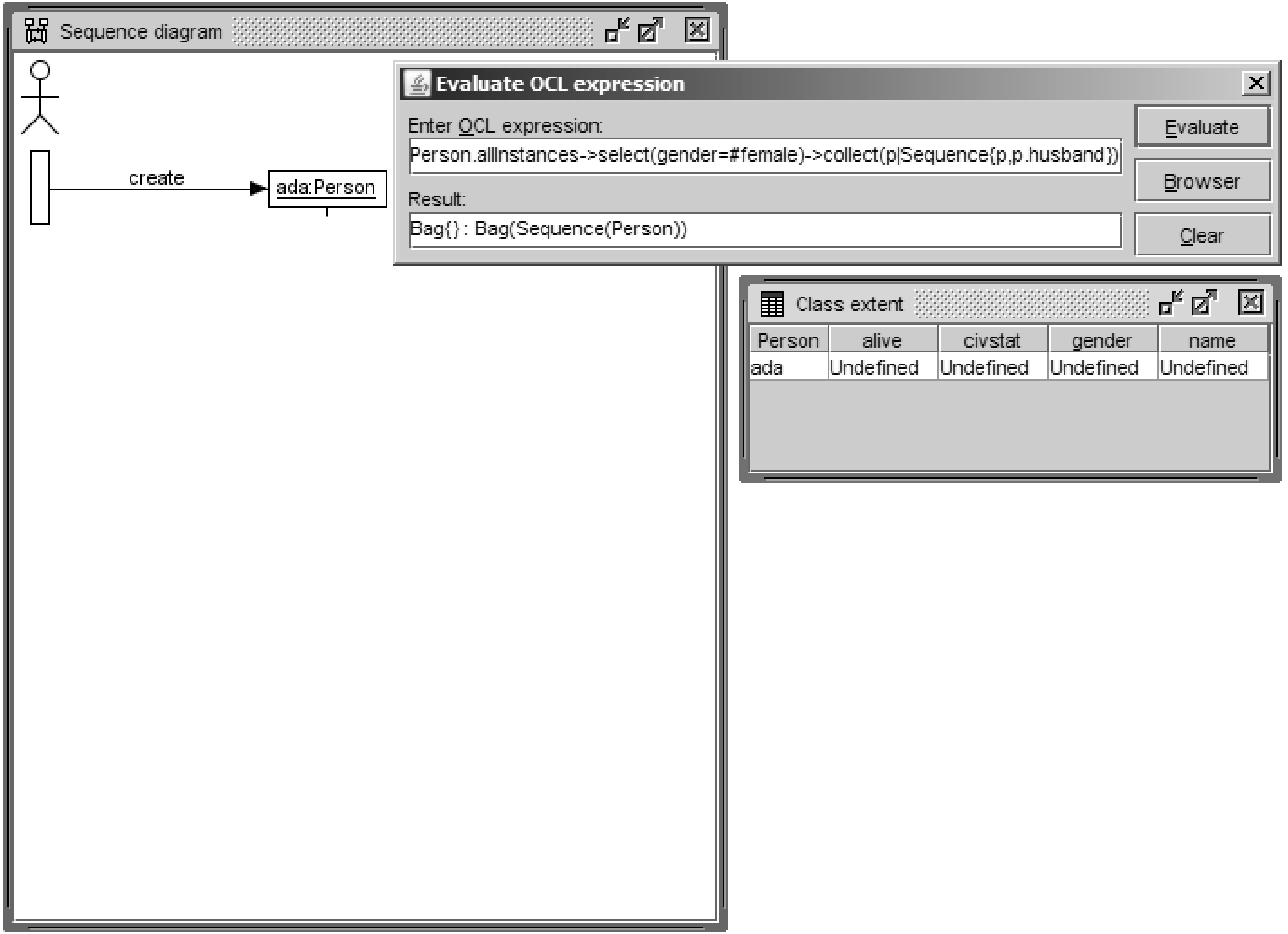
| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #married | #female | true |
| bob | 'Bob' | #divorced | #male | true |
| cyd | 'Cyd' | #married | #male | true |
| dan | 'Dan' | #single | #male | true |

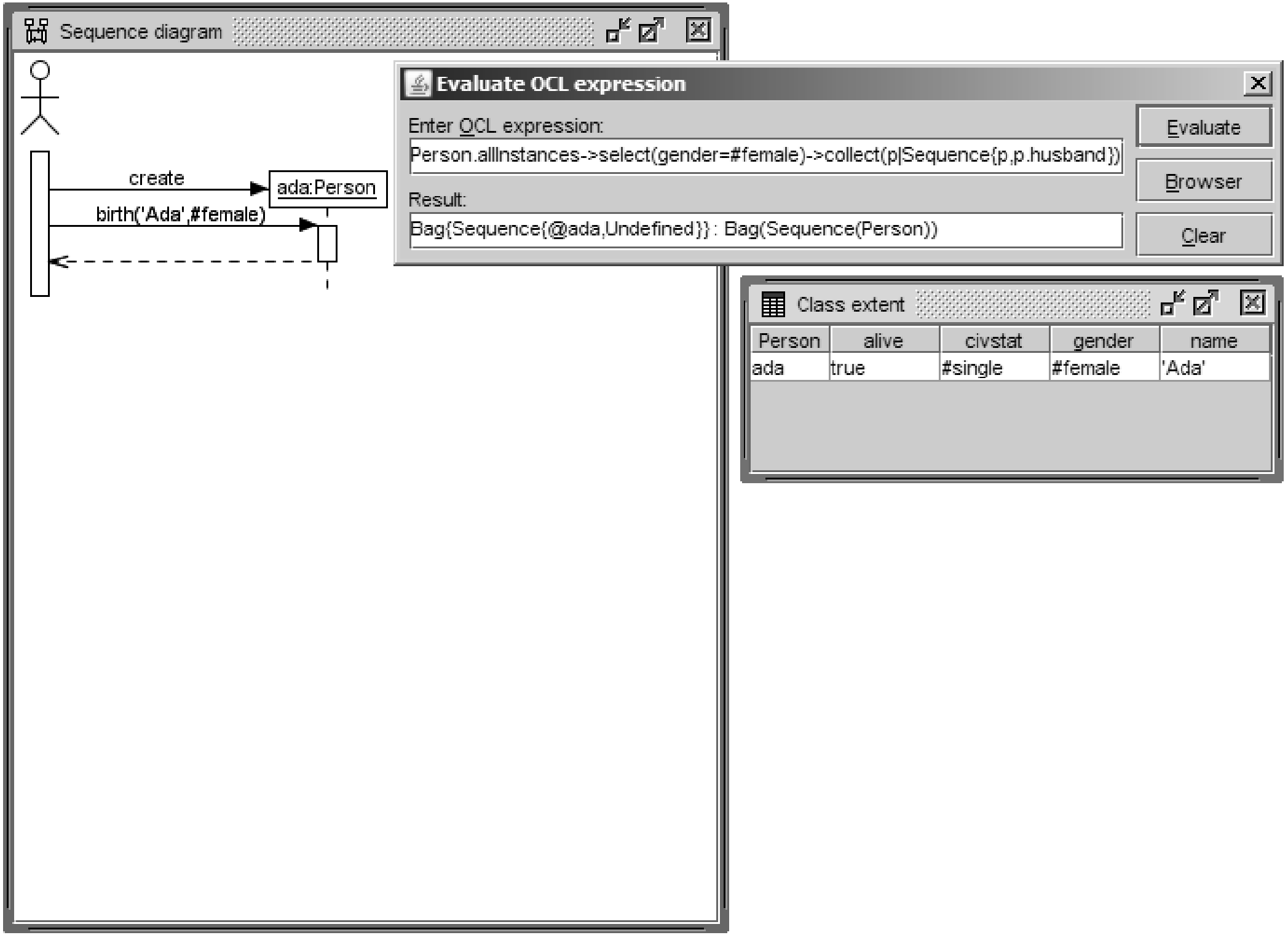


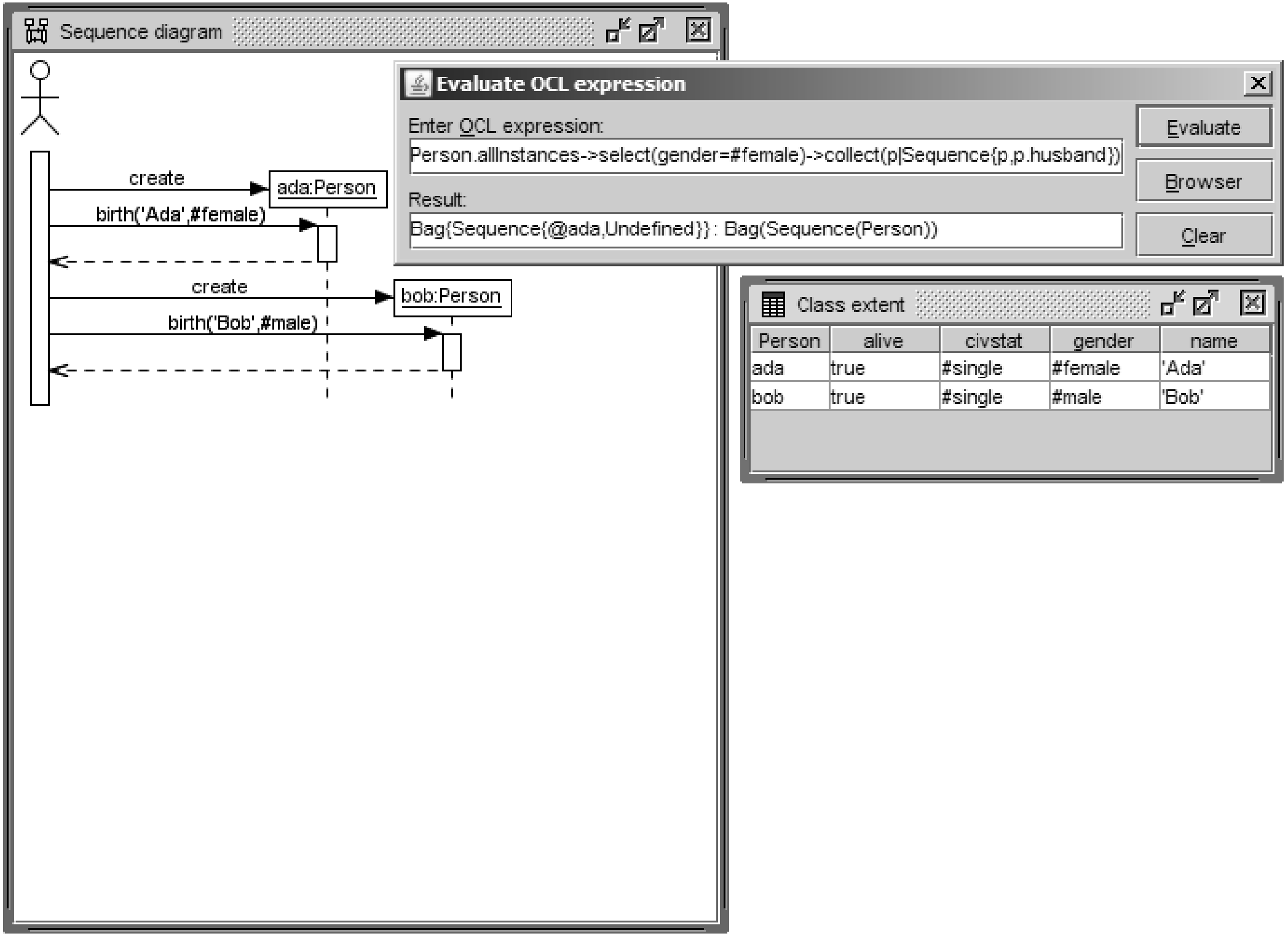



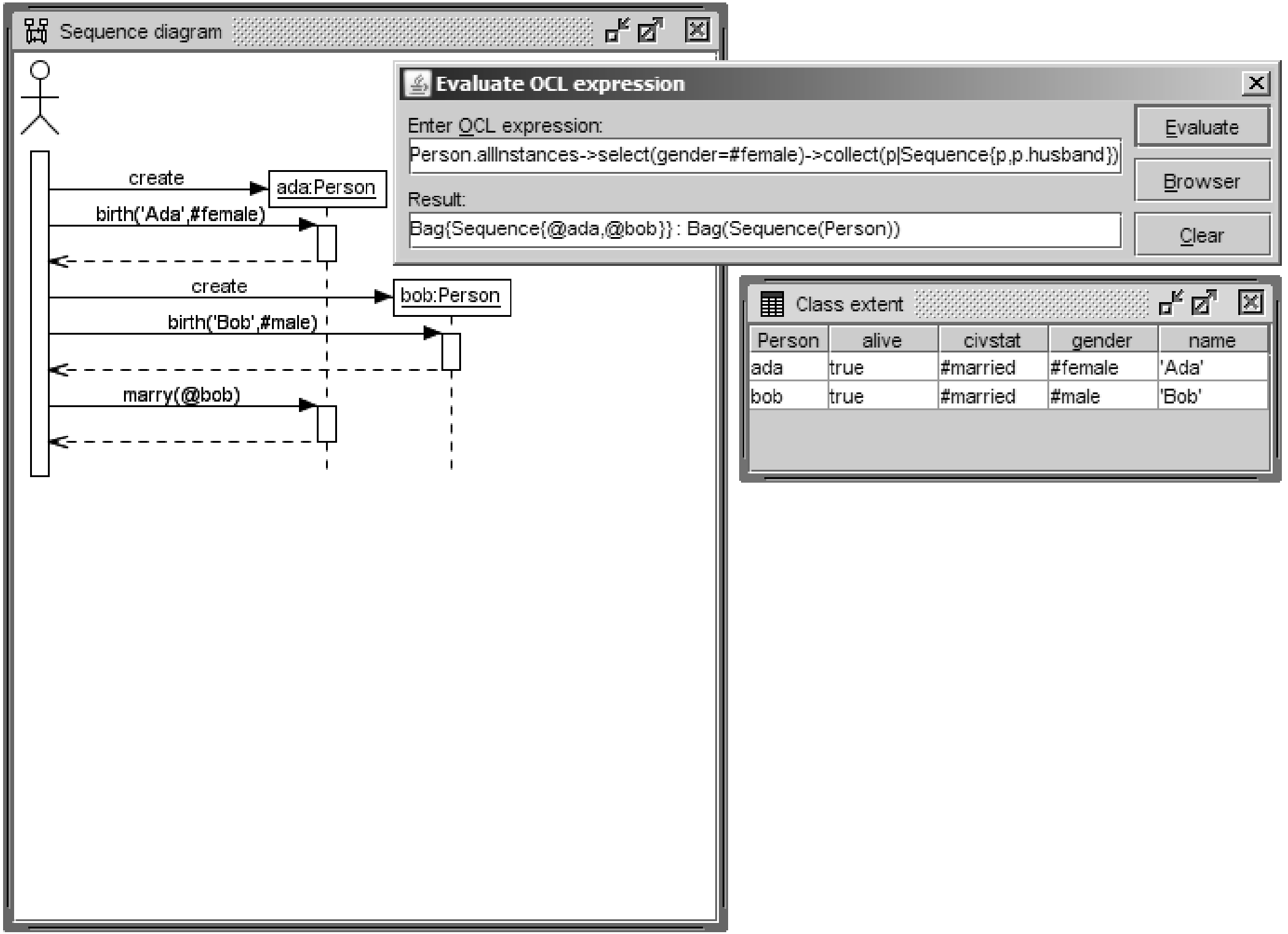

Class extent

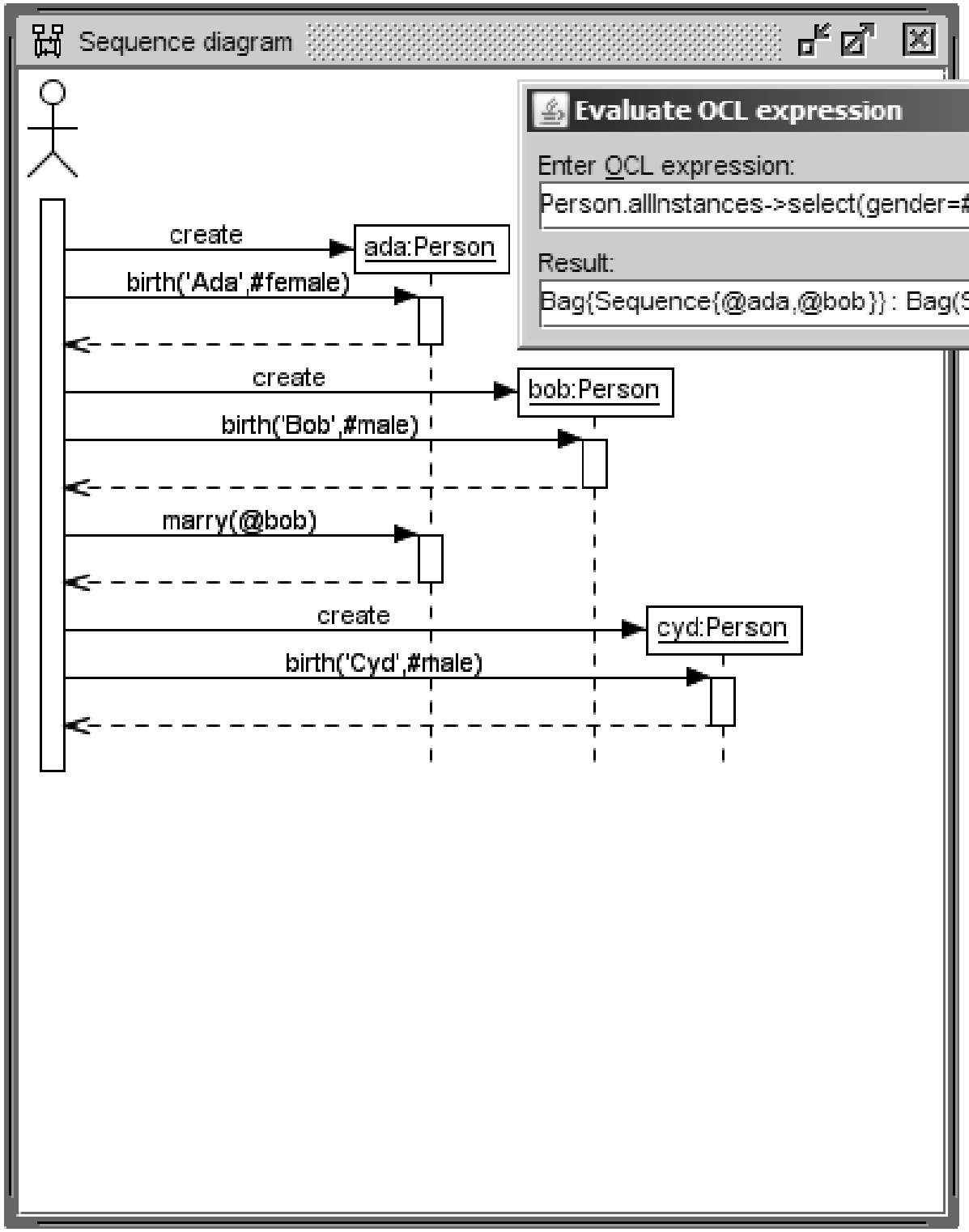
| Person | name | civstat | gender | alive |
|---|---|---|---|---|
| ada | 'Ada' | #widowed | #female | true |
| bob | 'Bob' | #divorced | #male | true |
| cyd | 'Cyd' | #married | #male | false |
| dan | 'Dan' | #single | #male | true |


Sequence diagram
create
ada:Person
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
Result:
Bag{} : Bag(Sequence(Person))
Evaluate
Browser
Clear
Class extent
Person
alive
civstat
gender
name
ada
Undefined
Undefined
Undefined
Undefined


Sequence diagram
create
ada:Person
birth('Ada',#female)
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
Result:
Bag{Sequence{@ada,Undefined}} : Bag(Sequence(Person))
Evaluate
Browser
Clear
Class extent
Person
alive
civstat
gender
name
ada
true
#single
#female
'Ada'


Sequence diagram
create
ada:Person
birth('Ada',#female)
create
bob:Person
birth('Bob',#male)
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
Result:
Bag{Sequence{@ada,Undefined}} : Bag(Sequence(Person))
Evaluate
Browser
Clear
Class extent
Person
alive
civstat
gender
name
ada
true
#single
#female
'Ada'
bob
true
#single
#male
'Bob'


Sequence diagram
create
ada:Person
birth('Ada',#female)
create
bob:Person
birth('Bob',#male)
marry(@bob)
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
Result:
Bag{Sequence{@ada,@bob}} : Bag(Sequence(Person))
Evaluate
Browser
Clear
Class extent
Person
alive
civstat
gender
name
ada
true
#married
#female
'Ada'
bob
true
#married
#male
'Bob'




**Evaluate OCL expression**

Enter OCL expression:

```
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
```

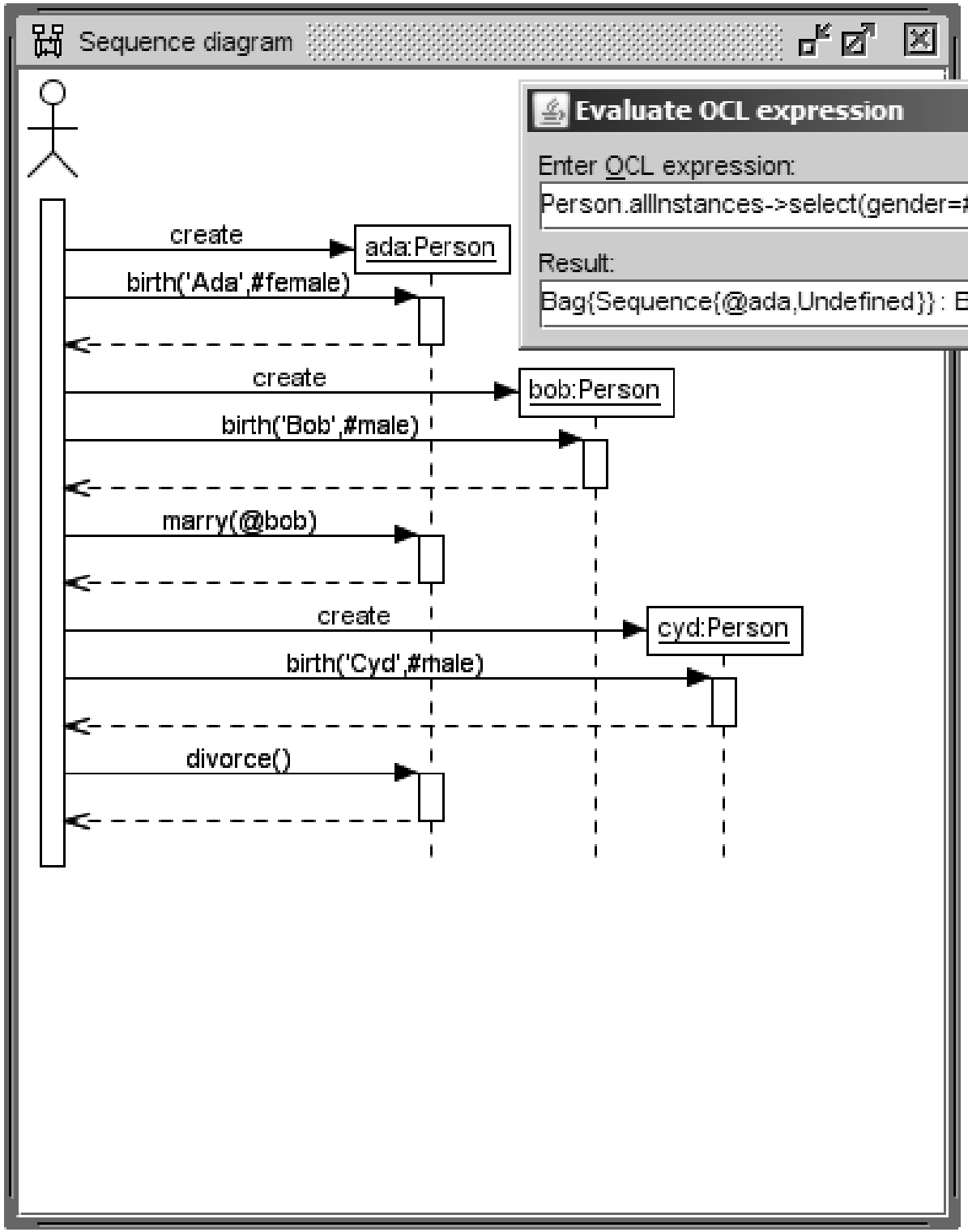
Result:

```
Bag{Sequence{@ada,@bob}} : Bag(Sequence(Person))
```

Evaluate | Browser | Clear

**Class extent**

| Person | alive | civstat | gender | name |
|---|---|---|---|---|
| ada | true | #married | #female | 'Ada' |
| bob | true | #married | #male | 'Bob' |
| cyd | true | #single | #male | 'Cyd' |




**Evaluate OCL expression**

Enter OCL expression:

```
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
```

Evaluate

Browser

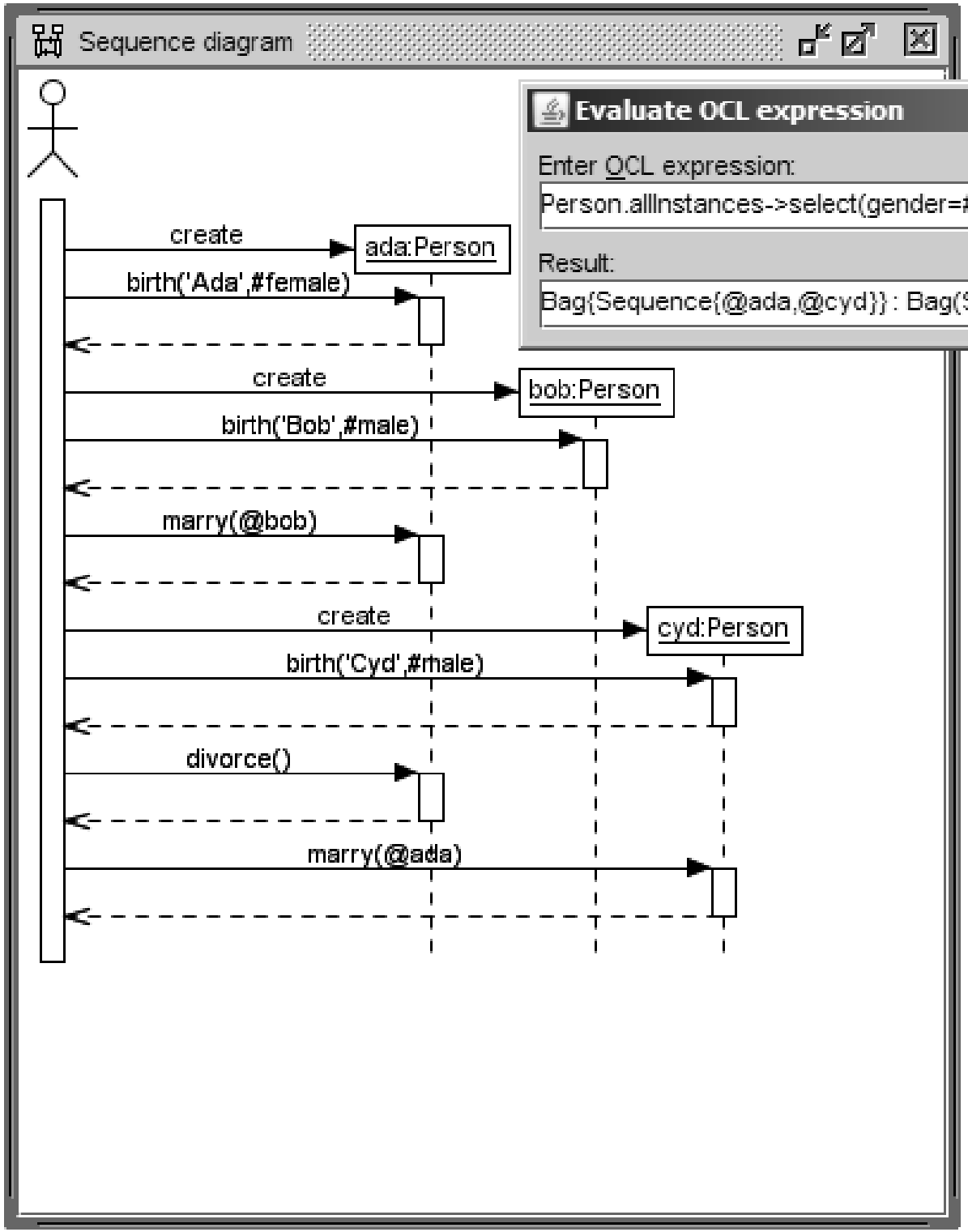
Clear

Result:

```
Bag{Sequence{@ada,Undefined}} : Bag(Sequence(Person))
```

**Class extent**

| Person | alive | civstat | gender | name |
|---|---|---|---|---|
| ada | true | #divorced | #female | 'Ada' |
| bob | true | #divorced | #male | 'Bob' |
| cyd | true | #single | #male | 'Cyd' |




**Evaluate OCL expression**

Enter OCL expression:

```
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
```

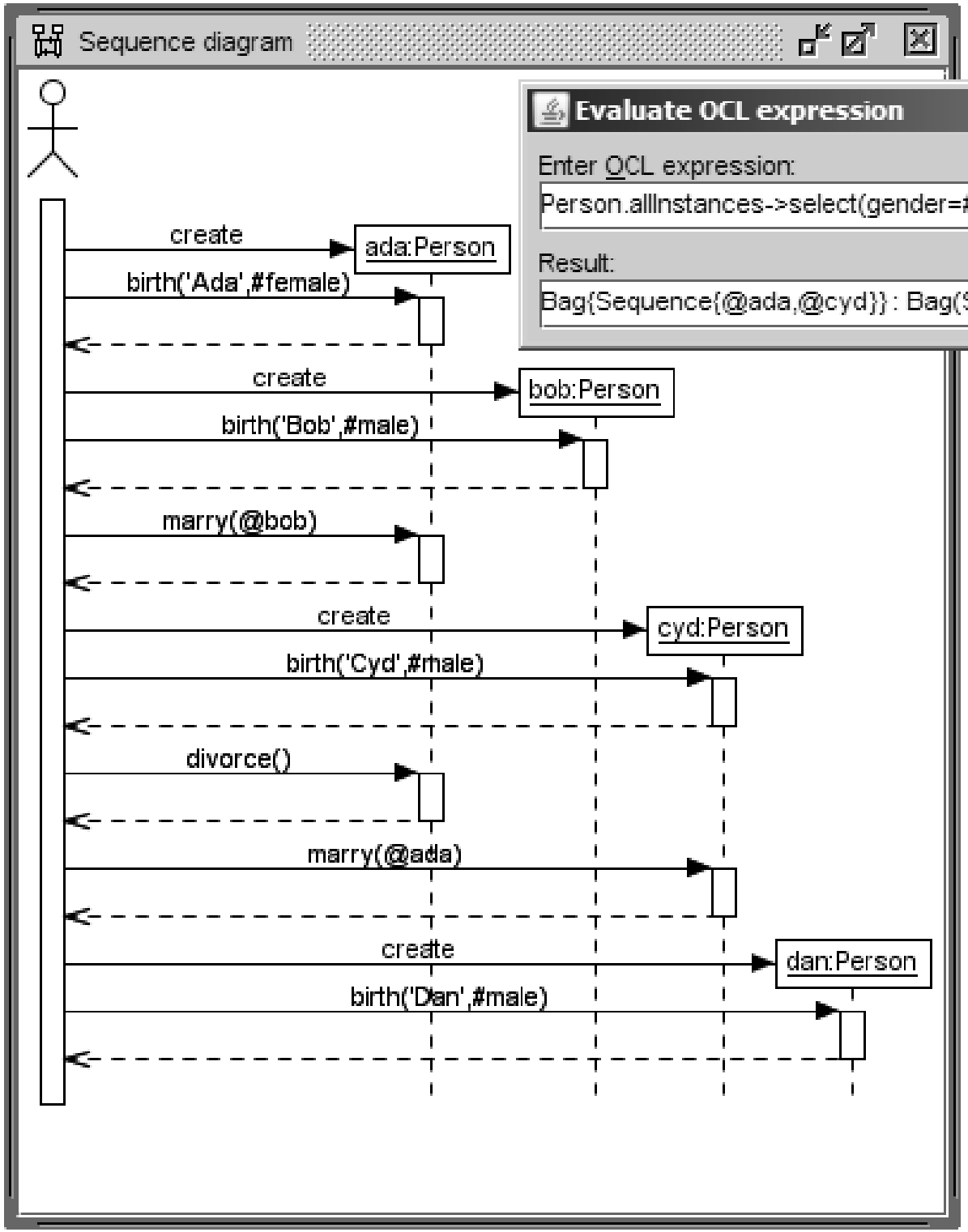
Result:

```
Bag{Sequence{@ada,@cyd}} : Bag(Sequence(Person))
```

Evaluate | Browser | Clear

**Class extent**

| Person | alive | civstat | gender | name |
|---|---|---|---|---|
| ada | true | #married | #female | 'Ada' |
| bob | true | #divorced | #male | 'Bob' |
| cyd | true | #married | #male | 'Cyd' |




**Evaluate OCL expression**

Enter OCL expression:

```
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
```

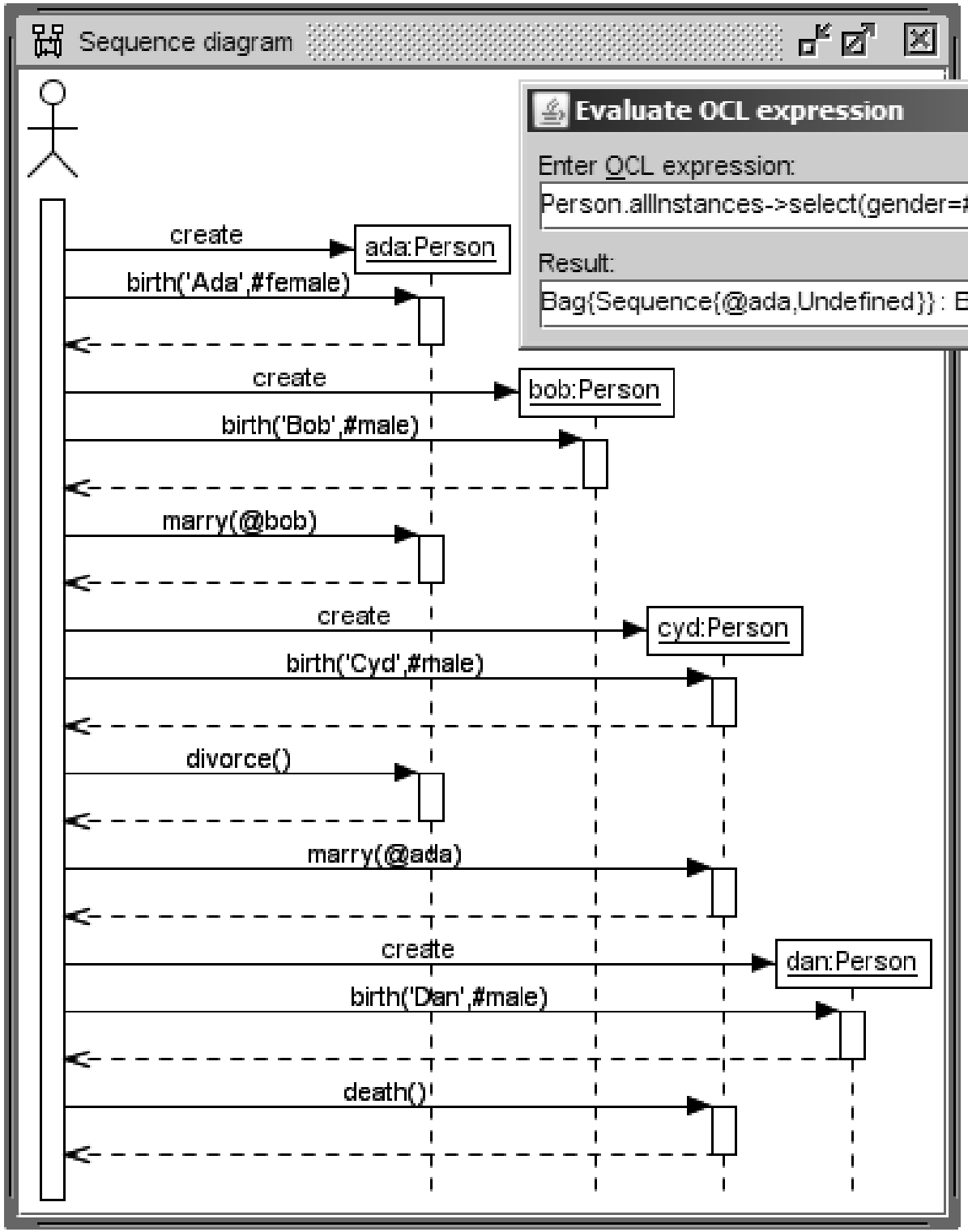
Result:

```
Bag{Sequence{@ada,@cyd}} : Bag(Sequence(Person))
```

Evaluate | Browser | Clear

Class extent

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




**Evaluate OCL expression**

Enter OCL expression:

```
Person.allInstances->select(gender=#female)->collect(p|Sequence{p,p.husband})
```

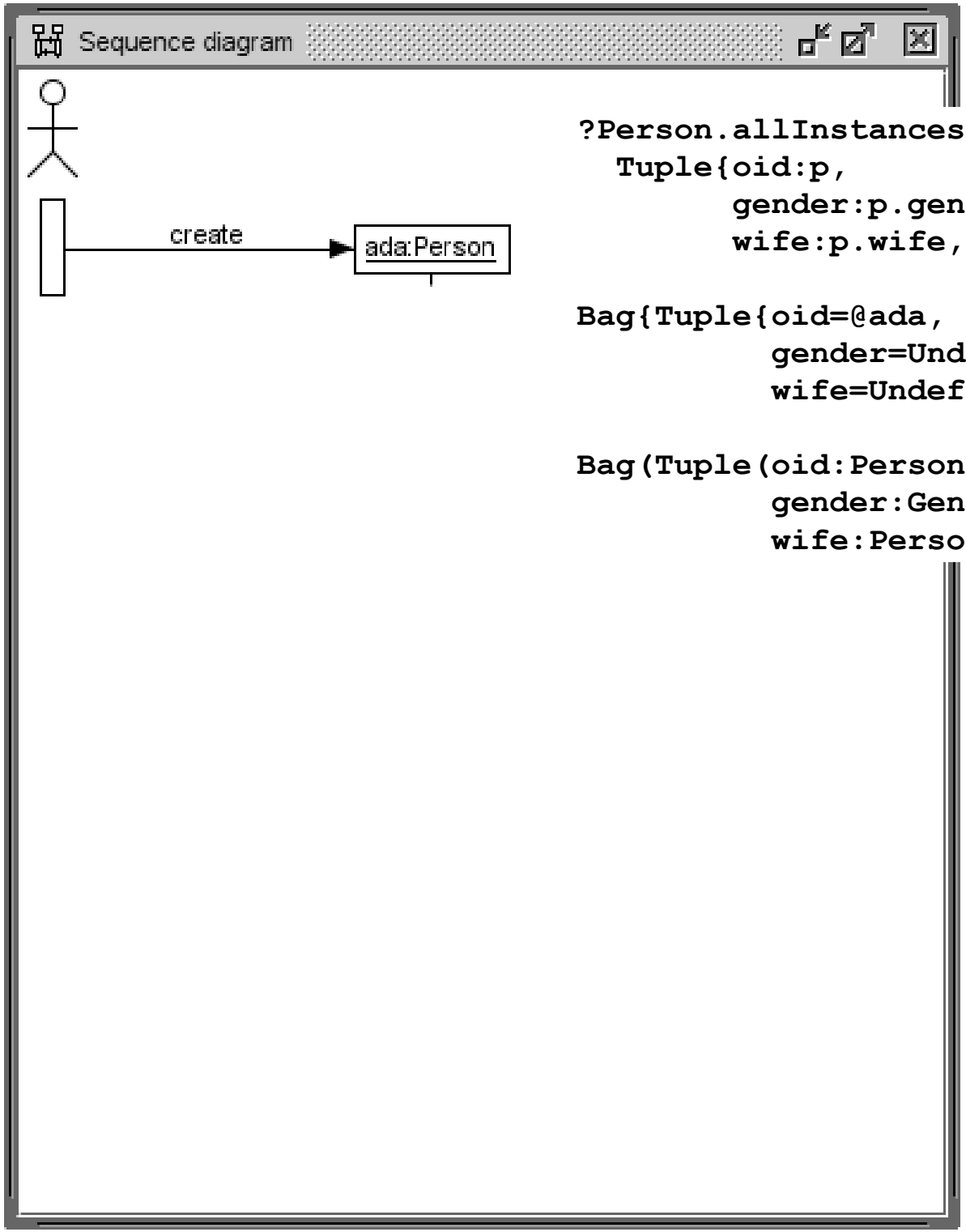
Result:

```
Bag{Sequence{@ada,Undefined}} : Bag(Sequence(Person))
```

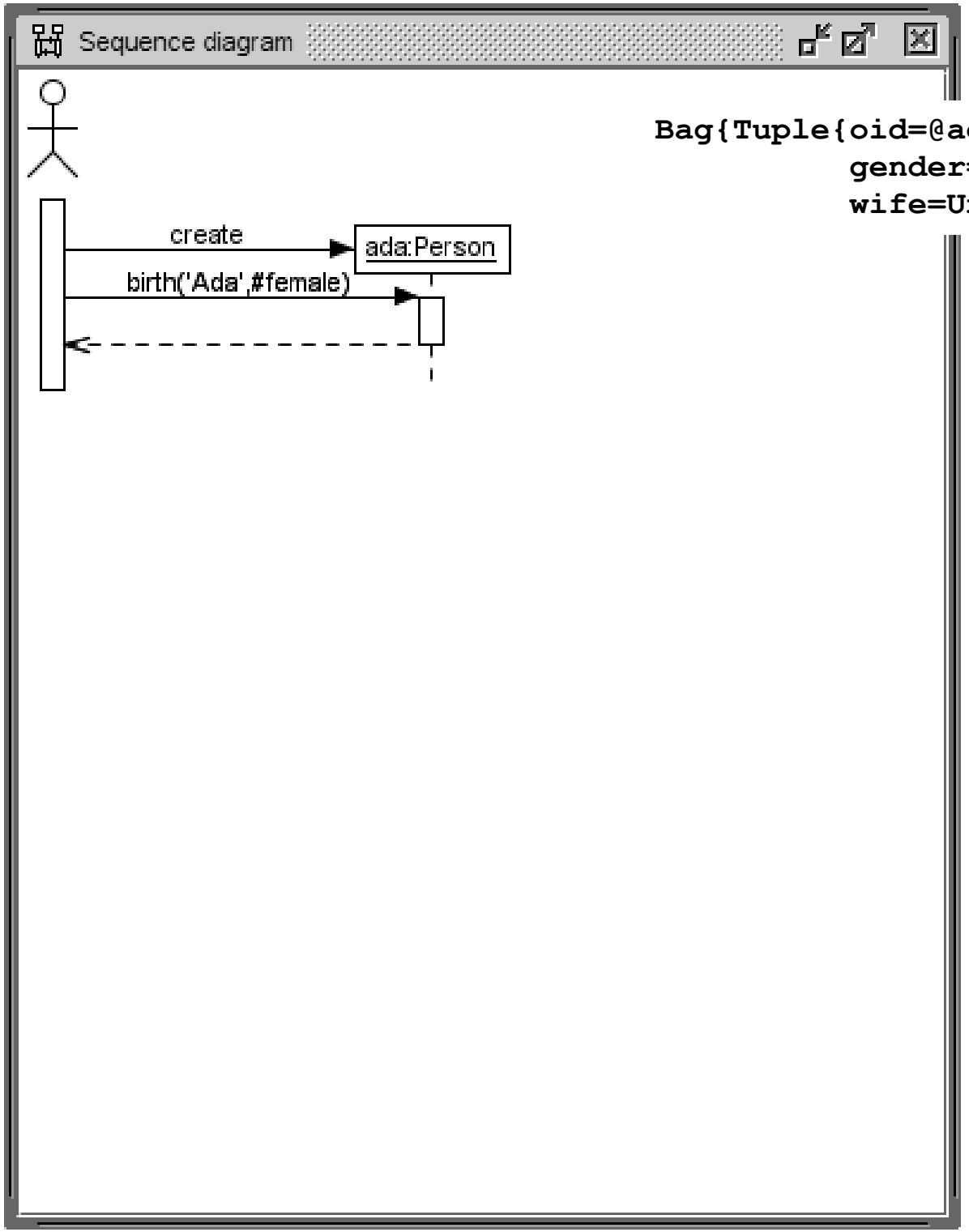
Evaluate | Browser | Clear

Class extent

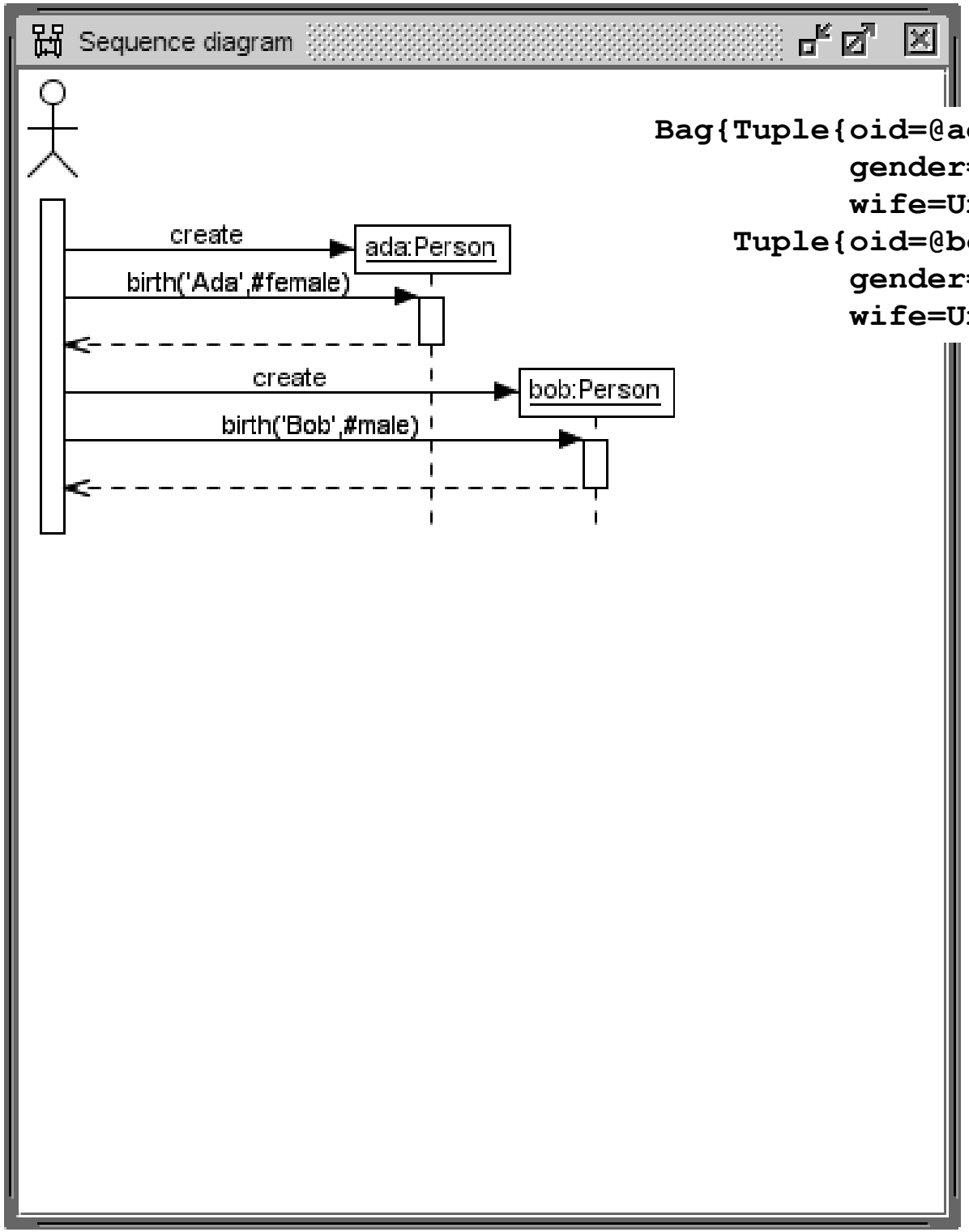
| Person | alive | civstat | gender | name |
|---|---|---|---|---|
| ada | true | #widowed | #female | 'Ada' |
| bob | true | #divorced | #male | 'Bob' |
| cyd | false | #married | #male | 'Cyd' |
| dan | true | #single | #male | 'Dan' |



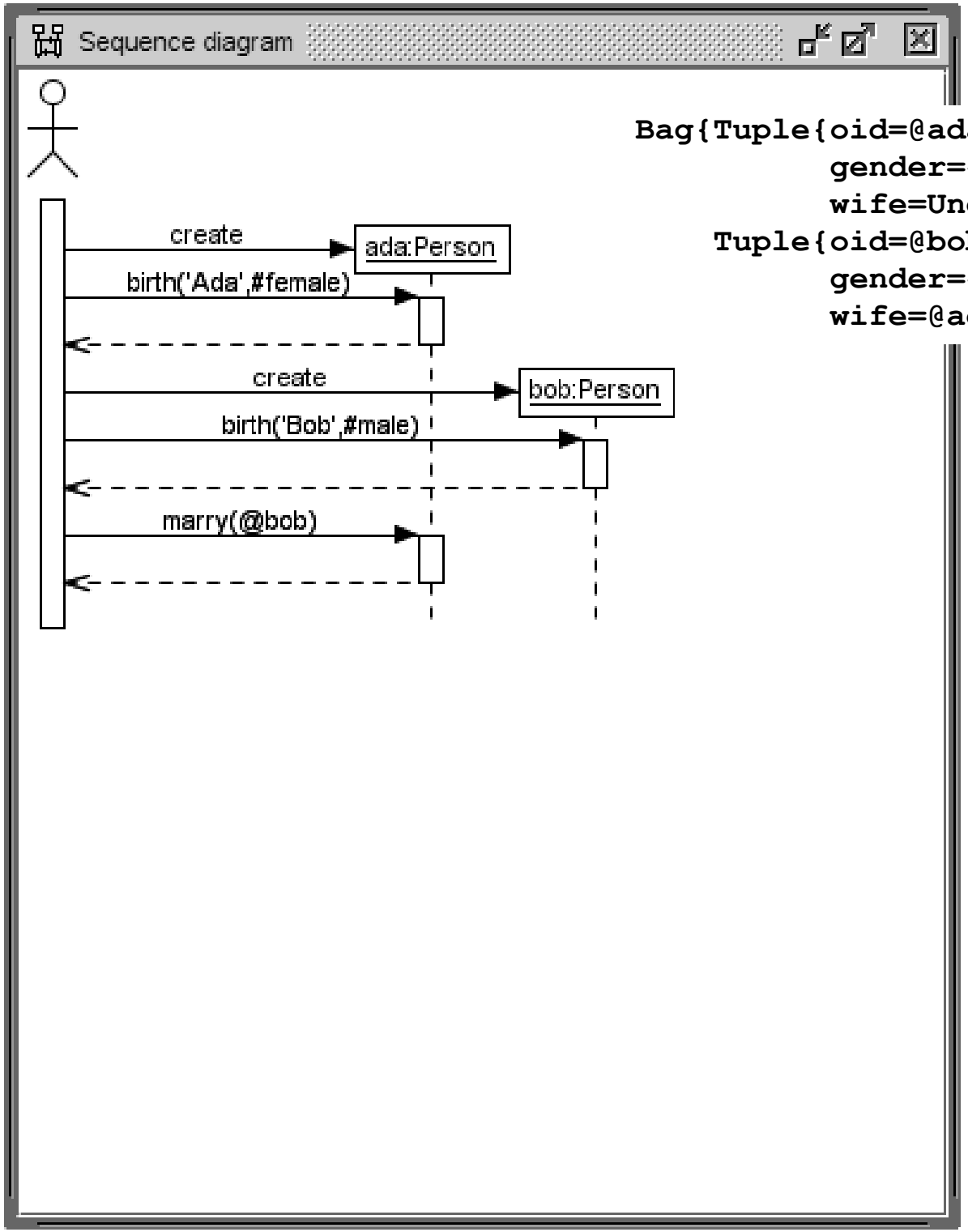

```
?Person.allInstances->collect(p|
  Tuple{oid:p,            name:p.name, civstat:p.civstat,
        gender:p.gender, alive:p.alive,
        wife:p.wife,      husband:p.husband})

Bag{Tuple{oid=@ada,      name=Undef, civstat=Undef,
          gender=Undef, alive=Undef,
          wife=Undef,    husband=Undef}} :

Bag(Tuple(oid:Person,     name:String, civstat:CivilStatus,
          gender:Gender, alive:Boolean,
          wife:Person,    husband:Person))
```



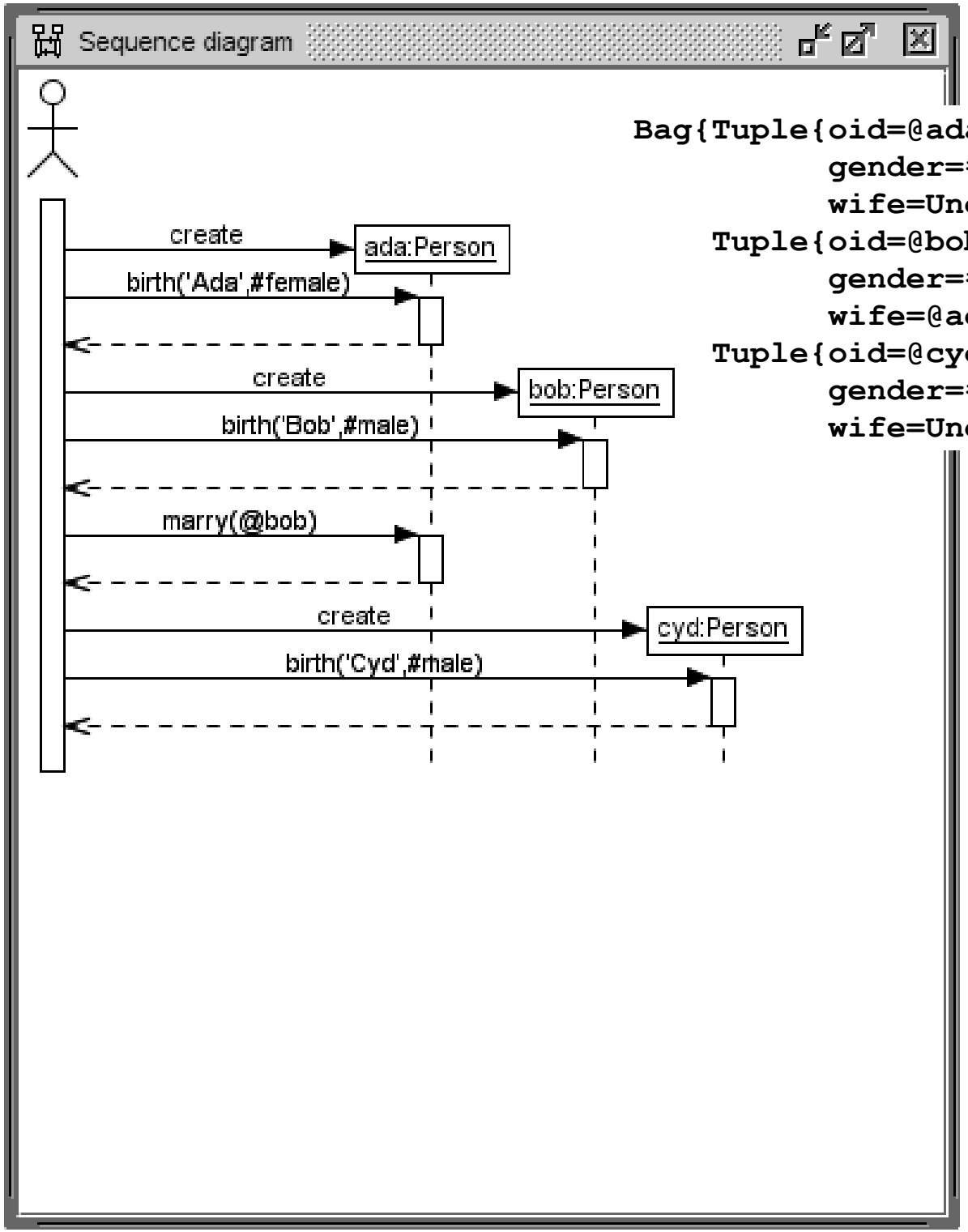

```
Bag{Tuple{oid=@ada,        name='Ada', civstat=#single,
          gender=#female, alive=true,
          wife=Undef,      husband=Undef}}
```



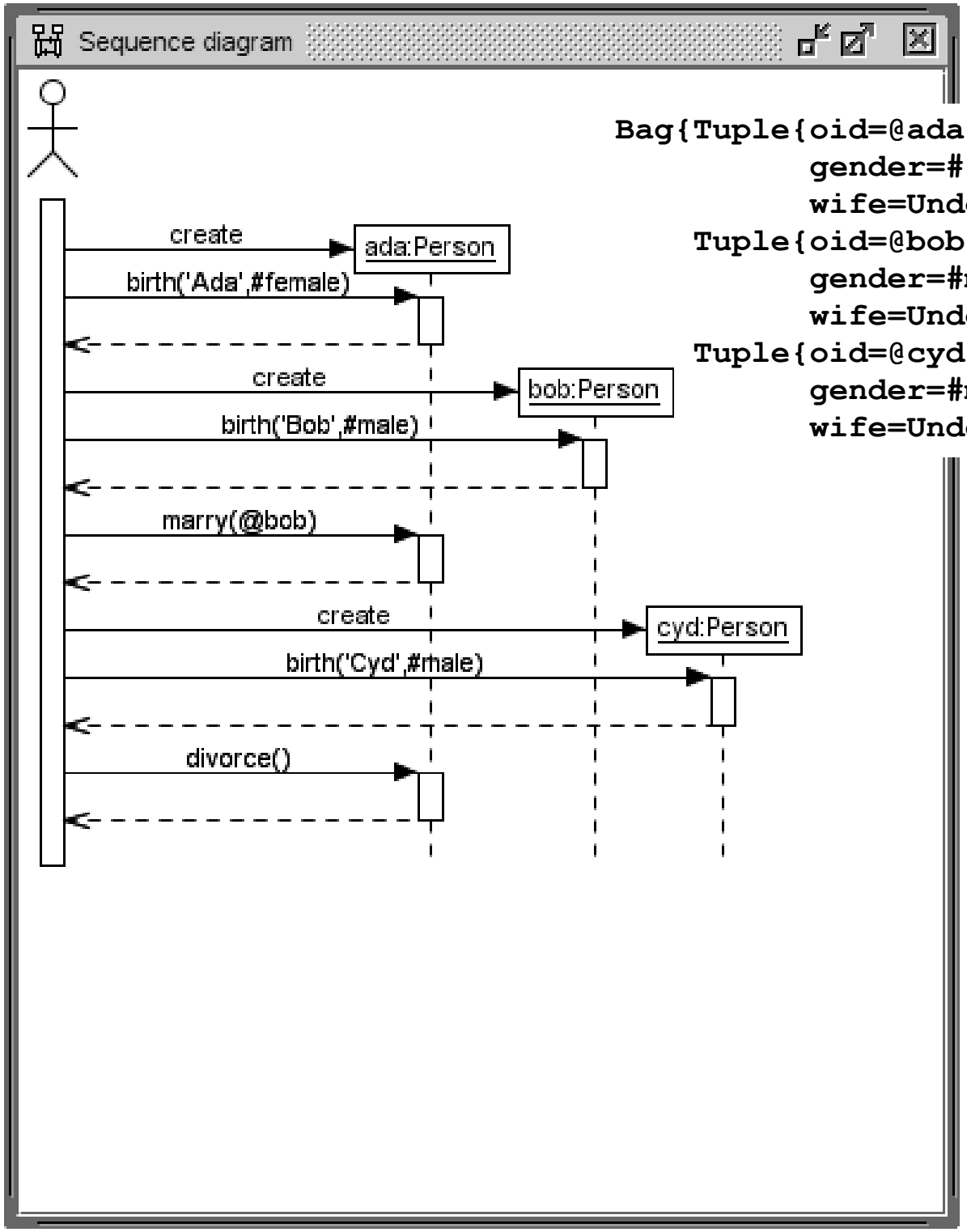

```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#single,
          gender=#female, alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@bob,       name='Bob', civstat=#single,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef}}
```



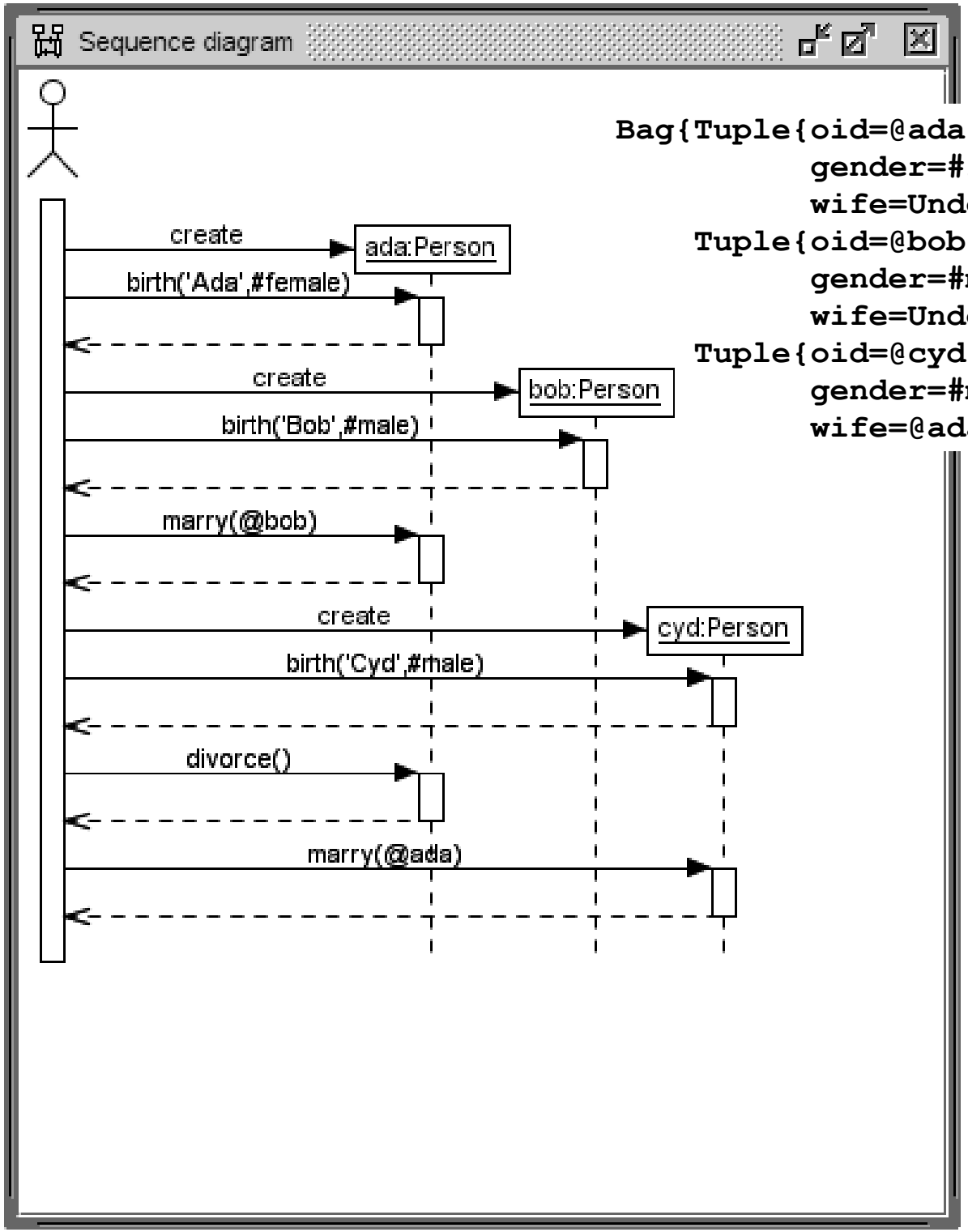

```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#married,
          gender=#female, alive=true,
          wife=Undef,     husband=@bob},
    Tuple{oid=@bob,       name='Bob', civstat=#married,
          gender=#male,   alive=true,
          wife=@ada,      husband=Undef}}
```



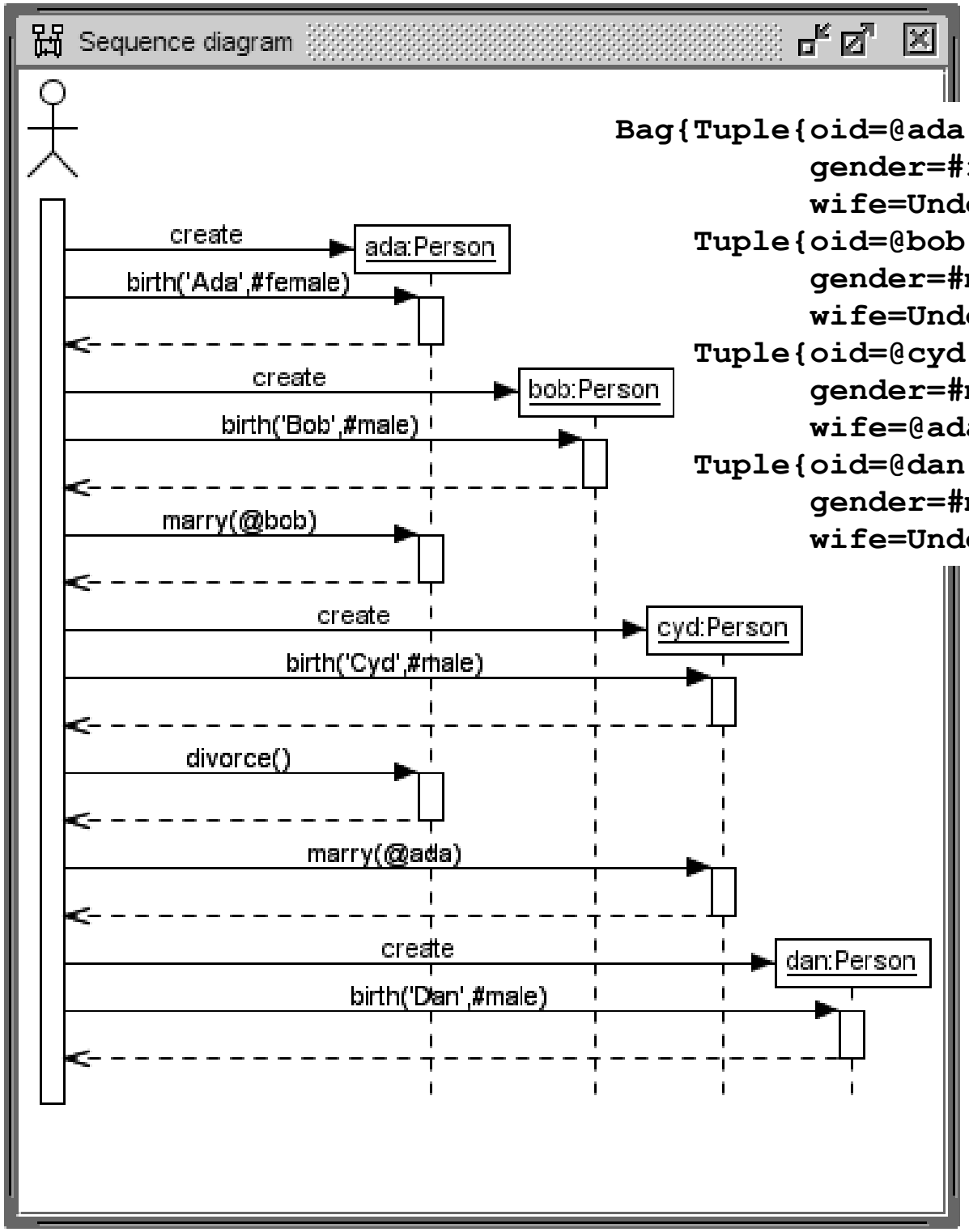

```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#married,
          gender=#female, alive=true,
          wife=Undef,     husband=@bob},
    Tuple{oid=@bob,       name='Bob', civstat=#married,
          gender=#male,   alive=true,
          wife=@ada,      husband=Undef},
    Tuple{oid=@cyd,       name='Cyd', civstat=#single,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef}}
```



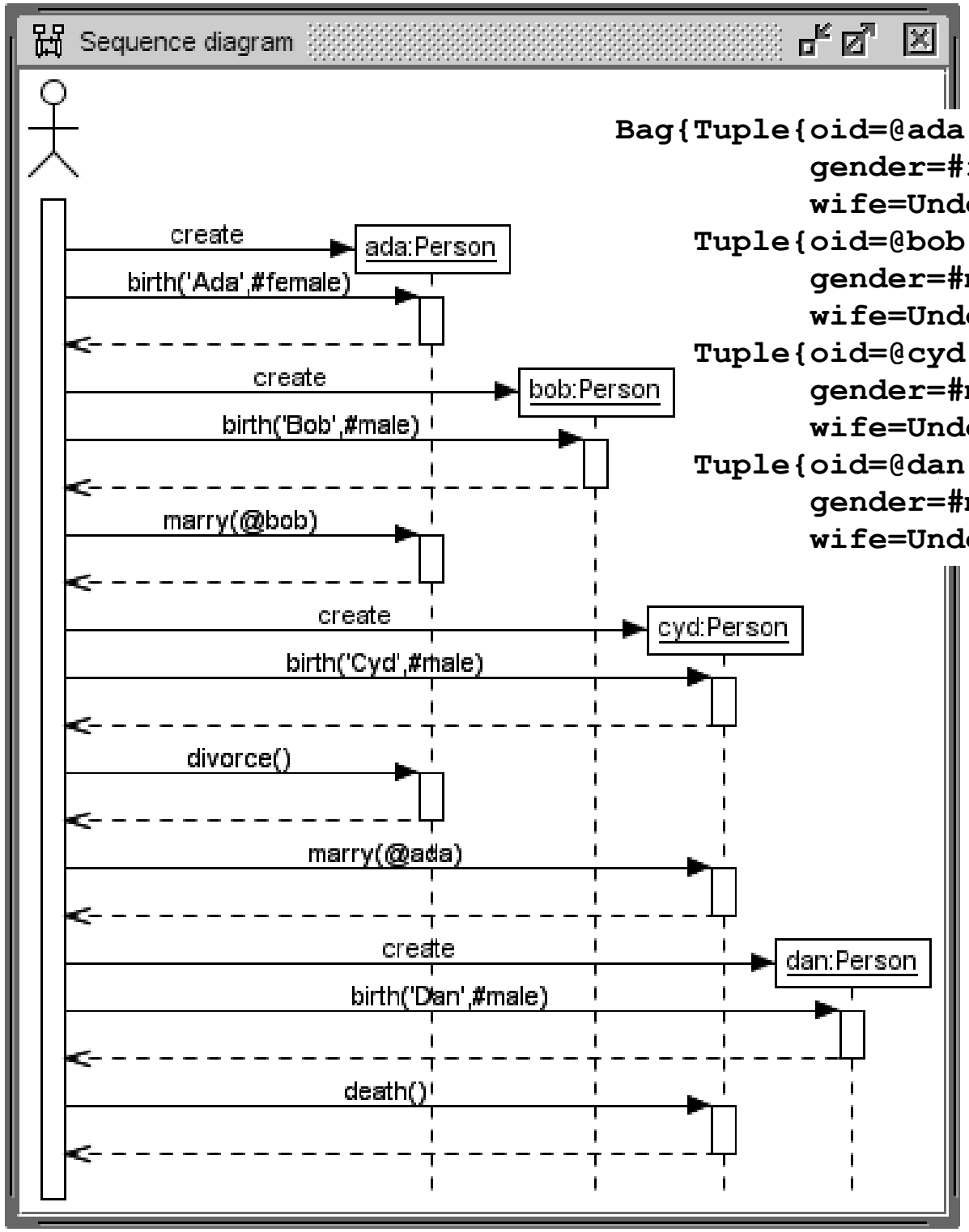

```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#divorced,
          gender=#female, alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@bob,       name='Bob', civstat=#divorced,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@cyd,       name='Cyd', civstat=#single,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef}}
```




```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#married,
          gender=#female, alive=true,
          wife=Undef,     husband=@cyd},
    Tuple{oid=@bob,       name='Bob', civstat=#divorced,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@cyd,       name='Cyd', civstat=#married,
          gender=#male,   alive=true,
          wife=@ada,      husband=Undef}}
```




```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#married,
          gender=#female, alive=true,
          wife=Undef,     husband=@cyd},
    Tuple{oid=@bob,       name='Bob', civstat=#divorced,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@cyd,       name='Cyd', civstat=#married,
          gender=#male,   alive=true,
          wife=@ada,      husband=Undef},
    Tuple{oid=@dan,       name='Dan', civstat=#single,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef}}
```




```
Bag{Tuple{oid=@ada,       name='Ada', civstat=#widowed,
          gender=#female, alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@bob,       name='Bob', civstat=#divorced,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef},
    Tuple{oid=@cyd,       name='Cyd', civstat=#married,
          gender=#male,   alive=false,
          wife=Undef,     husband=Undef},
    Tuple{oid=@dan,       name='Dan', civstat=#single,
          gender=#male,   alive=true,
          wife=Undef,     husband=Undef}}
```

```
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
```

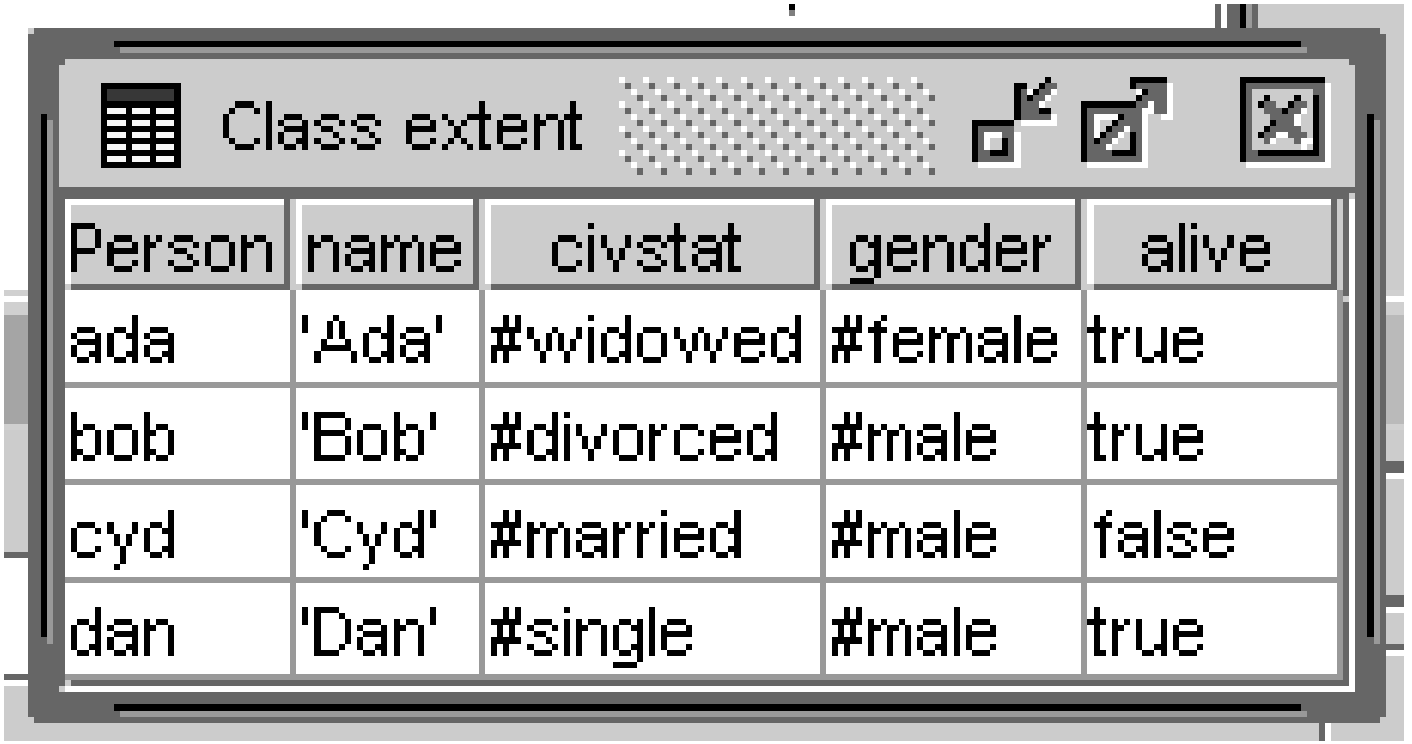
```
use> !openter ada divorce()
     precondition `isMarried' is true       precondition `isAlive' is true
     precondition `husbandAlive' is true    precondition `wifeAlive' is true
use> read Person_divorce.cmd
     Person_divorce.cmd> -- Person::divorce()
     Person_divorce.cmd> !set self.civstat:=#divorced
     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
```

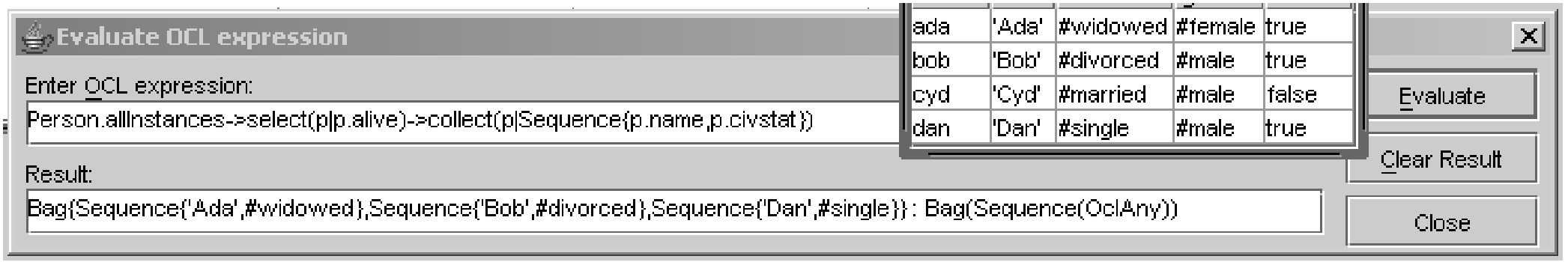
```
use> !create dan:Person

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() -- 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
```

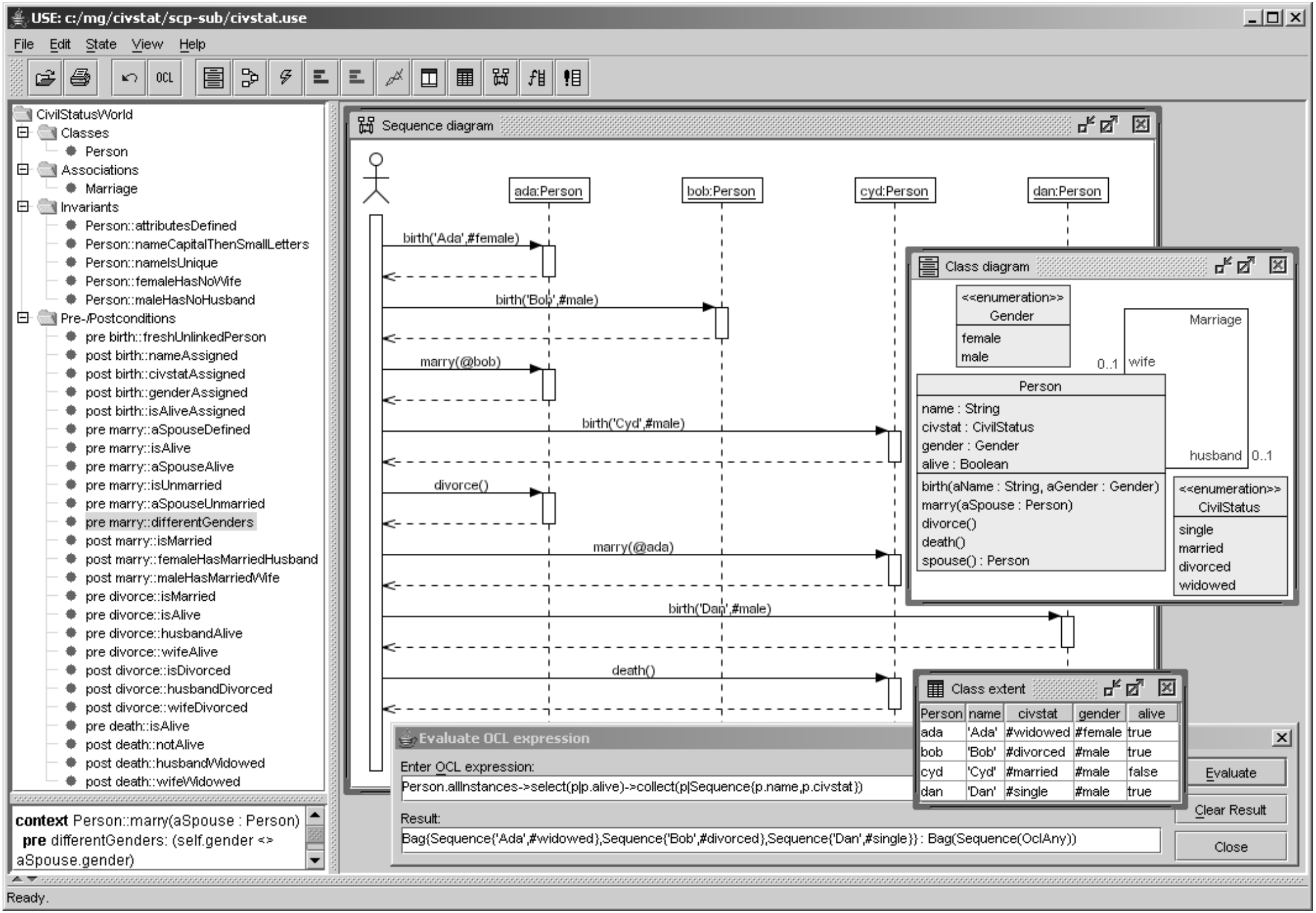
```
procedure Person_marry(self:Person,aSpouse:Person)
begin
[self].civstat:=[#married];
[aSpouse].civstat:=[#married];
if [self.gender=#female] then
  begin Insert(Marriage,[self],[aSpouse]); end
else -- [self.gender=#male]
  begin Insert(Marriage,[aSpouse],[self]); end;
end;

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

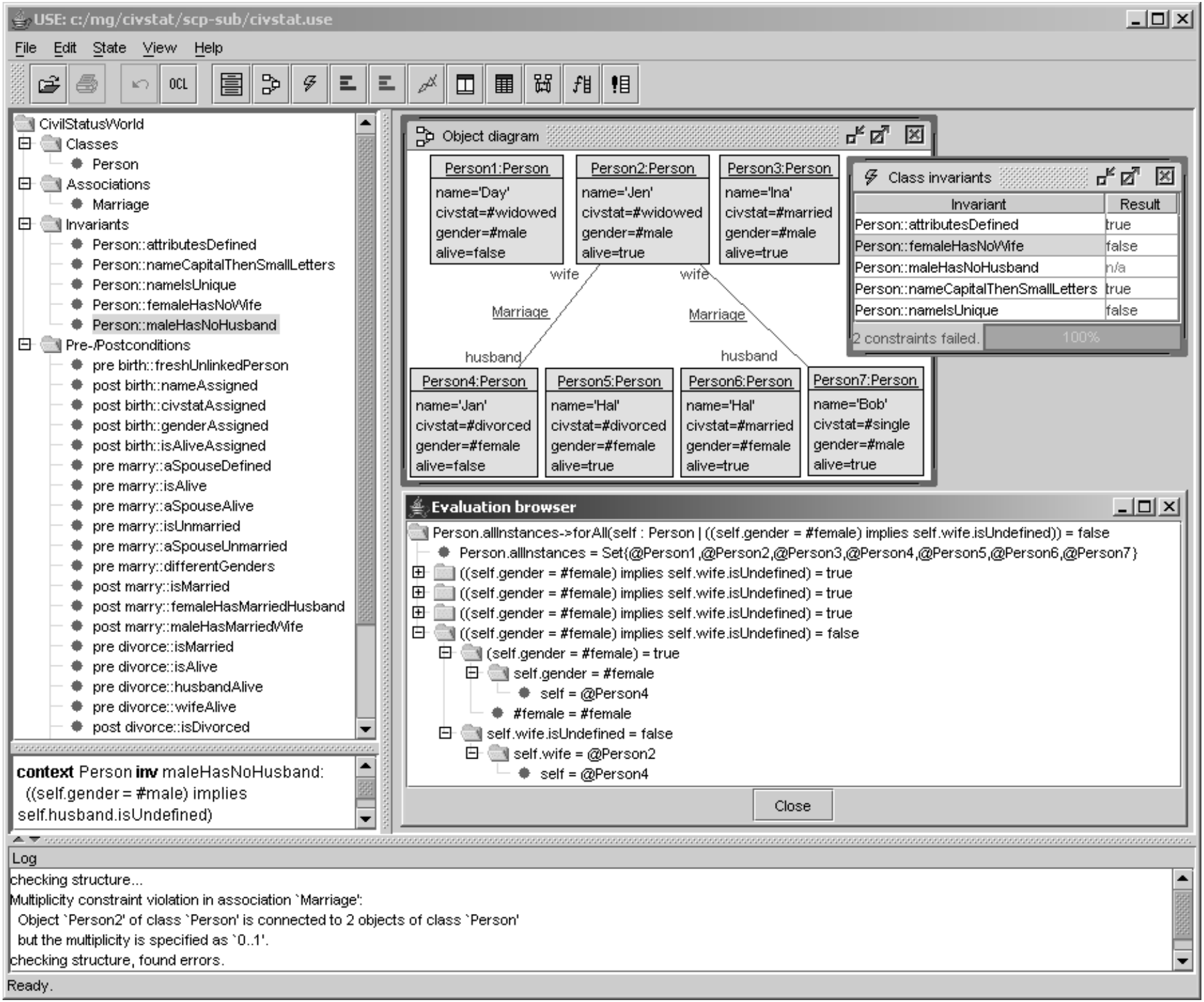
```
procedure Person_divorce(self:Person)
begin
[self].civstat:=[#divorced];
if [self.gender=#female] then
  begin [self.husband].civstat:=[#divorced];
  Delete(Marriage,[self],[self.husband]); end
else -- [self.gender=#male]
  begin [self.wife].civstat:=[#divorced];
  Delete(Marriage,[self.wife],[self]);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

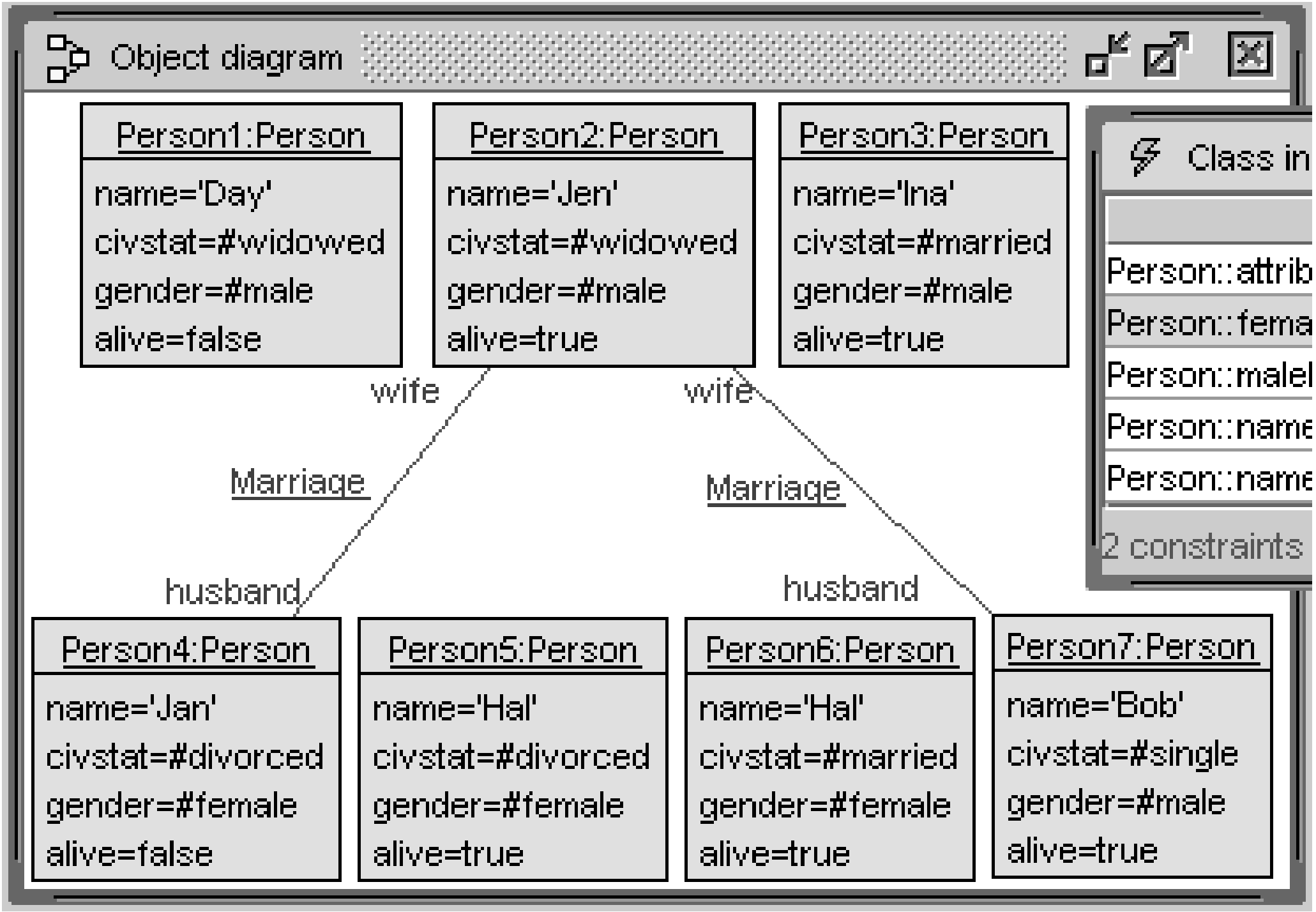
```
use> !openter ada marry(bob)
     precondition `aSpouseDefined' is true
     precondition `isAlive' is true
     precondition `aSpouseAlive' is true
     precondition `isUnmarried' is true
     precondition `aSpouseUnmarried' is true
     precondition `differentGenders' is true
use> gen start civstat.assl Person_marry(ada,bob)
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
```


Class extent
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


Evaluate OCL expression
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))
ada 'Ada' #widowed #female true
bob 'Bob' #divorced #male true
cyd 'Cyd' #married #male false
dan 'Dan' #single #male true
Evaluate
Clear Result
Close


USE: c:/mg/civstat/scp-sub/civstat.use
File Edit State View Help
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)
Sequence diagram
ada:Person
bob:Person
cyd:Person
dan:Person
birth('Ada',#female)
birth('Bob',#male)
marry(@bob)
birth('Cyd',#male)
divorce()
marry(@ada)
birth('Dan',#male)
death()
Class diagram
<<enumeration>>
Gender
female
male
Marriage
0..1
wife
husband
0..1
Person
name : String
civstat : CivilStatus
gender : Gender
alive : Boolean
birth(aName : String, aGender : Gender)
marry(aSpouse : Person)
divorce()
death()
spouse() : Person
<<enumeration>>
CivilStatus
single
married
divorced
widowed
Class extent
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
Evaluate OCL expression
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))
Evaluate
Clear Result
Close
Ready.


USE: c:/mg/civstat/scp-sub/civstat.use
File Edit State View Help
OCL
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)
Object diagram
Person1:Person
name='Day'
civstat=#widowed
gender=#male
alive=false
Person2:Person
name='Jen'
civstat=#widowed
gender=#male
alive=true
Person3:Person
name='Ina'
civstat=#married
gender=#male
alive=true
wife
wife
Marriage
Marriage
husband
husband
Person4:Person
name='Jan'
civstat=#divorced
gender=#female
alive=false
Person5:Person
name='Hal'
civstat=#divorced
gender=#female
alive=true
Person6:Person
name='Hal'
civstat=#married
gender=#female
alive=true
Person7:Person
name='Bob'
civstat=#single
gender=#male
alive=true
Class invariants
Invariant
Result
Person::attributesDefined
true
Person::femaleHasNoWife
false
Person::maleHasNoHusband
n/a
Person::nameCapitalThenSmallLetters
true
Person::nameIsUnique
false
2 constraints failed.
100%
Evaluation browser
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
Close
Log
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.
Ready.

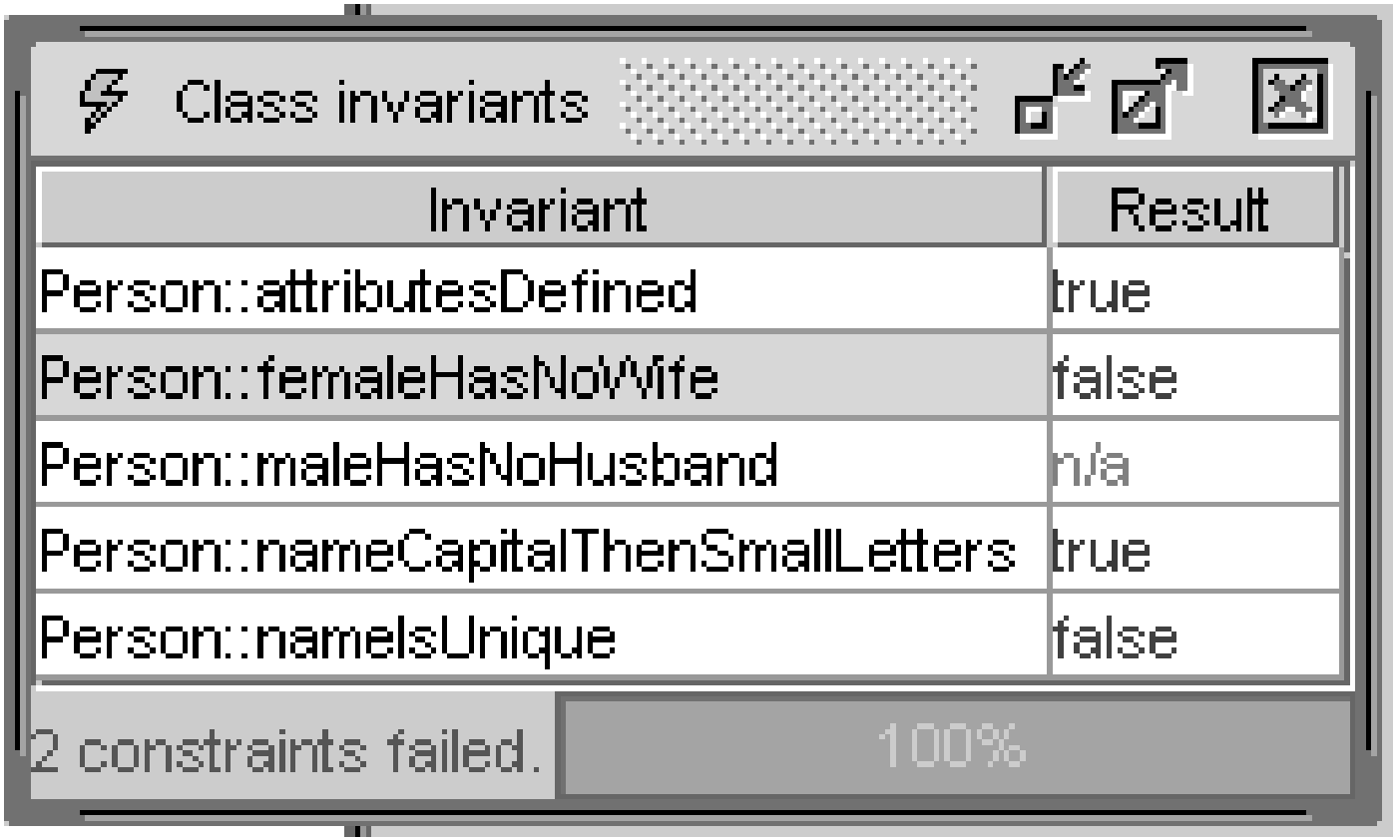

Object diagram
Person1:Person
name='Day'
civstat=#widowed
gender=#male
alive=false
Person2:Person
name='Jen'
civstat=#widowed
gender=#male
alive=true
Person3:Person
name='Ina'
civstat=#married
gender=#male
alive=true
Class in
Person::attrib
Person::fema
Person::malel
Person::name
Person::name
2 constraints
wife
wife
Marriage
Marriage
husband
husband
Person4:Person
name='Jan'
civstat=#divorced
gender=#female
alive=false
Person5:Person
name='Hal'
civstat=#divorced
gender=#female
alive=true
Person6:Person
name='Hal'
civstat=#married
gender=#female
alive=true
Person7:Person
name='Bob'
civstat=#single
gender=#male
alive=true

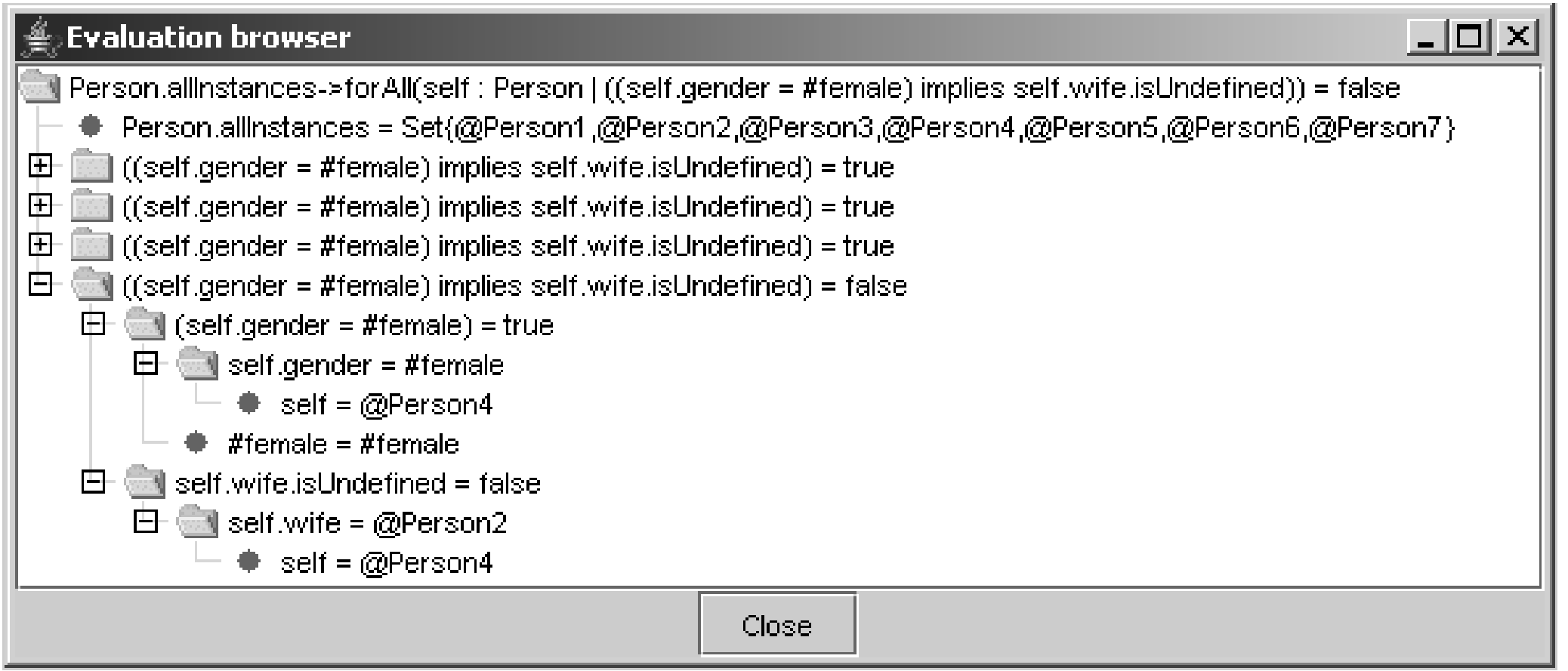
```
procedure crowd(numFem:Integer, numMale:Integer, numMarr:Integer)
var theFemales: Sequence(Person), theMales: Sequence(Person),
    f: Person, m: Person;
begin
  theFemales:=CreateN(Person,[numFem]);
  theMales:=CreateN(Person,[numMale]);
  for i:Integer in [Sequence{1..numFem}] begin
    [theFemales->at(i)].name:=Any([Sequence{'Ada','Bel','Cam',
      '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
     ... end;
  for i:Integer in [Sequence{1..numMarr}] begin
    f:=Any([theFemales]); m:=Any([theMales]);
    Insert(Marriage,[f],[m]);
    end;
end;
```

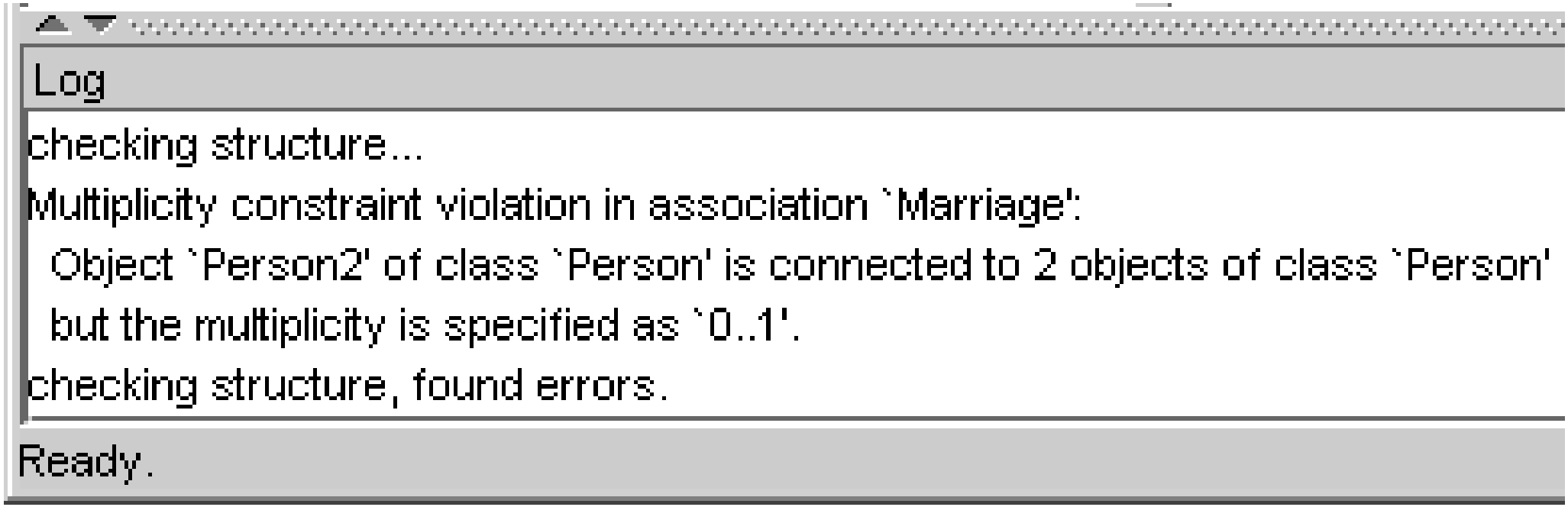
**crowd(3,4,2)**

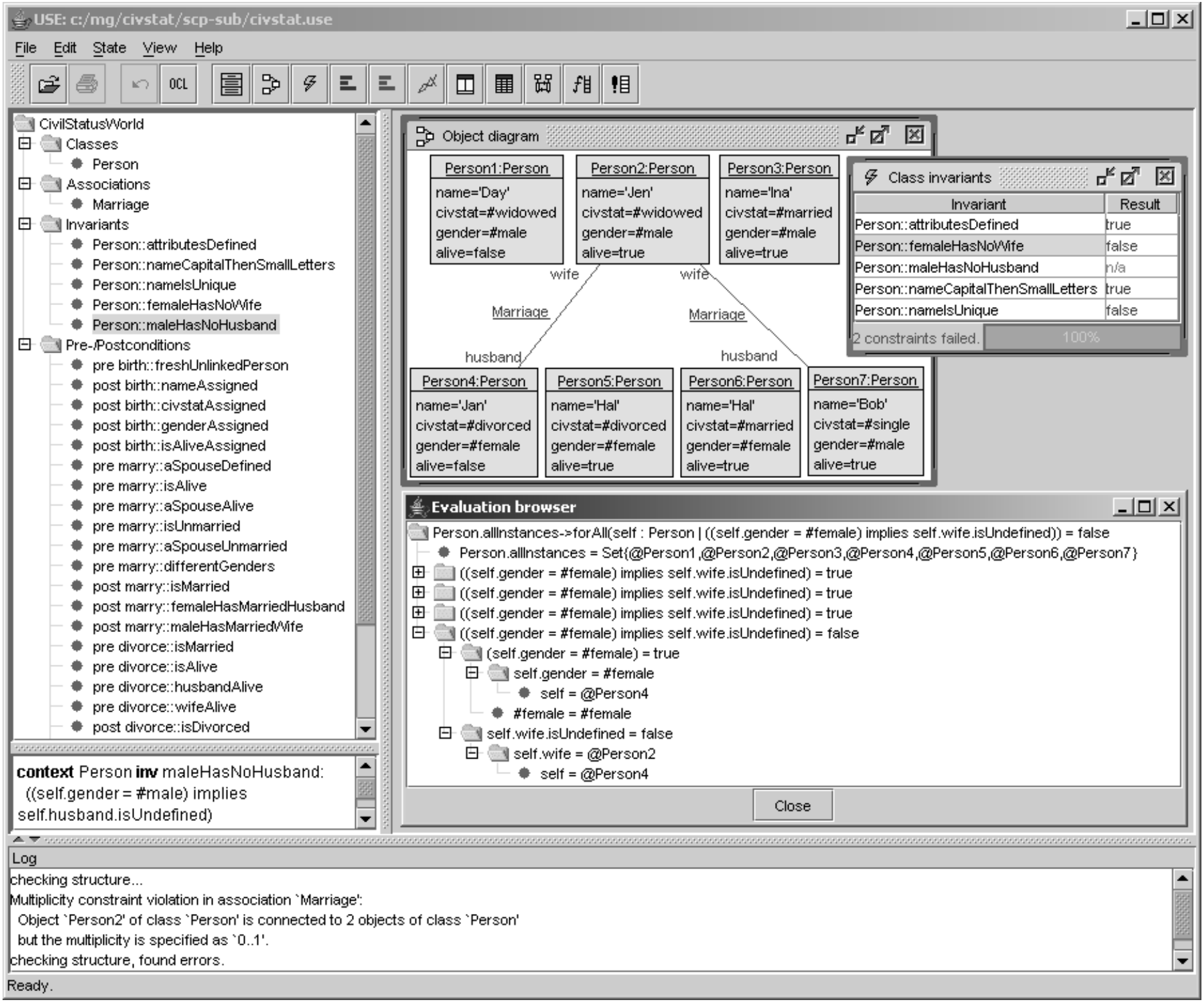
```
procedure crowd(numFemale:Integer, numMale:Integer, numMarriage:Integer)

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;
```

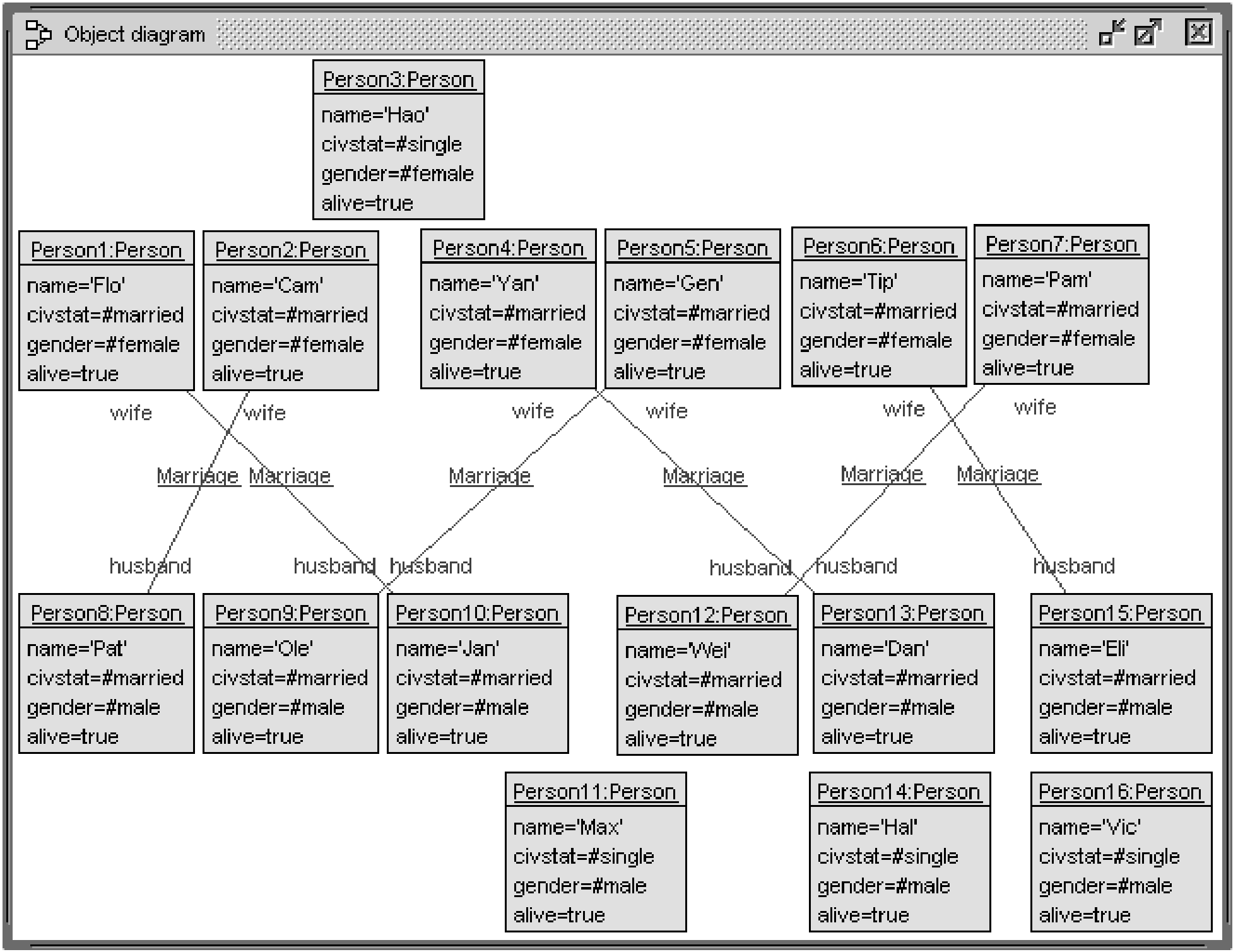
```
use> open civstat.use

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 ...
     !set ...
     !insert ...
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.
```


Class invariants
Invariant
Result
Person::attributesDefined
true
Person::femaleHasNoWife
false
Person::maleHasNoHusband
n/a
Person::nameCapitalThenSmallLetters
true
Person::nameIsUnique
false
2 constraints failed.
100%


Evaluation browser
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
Close

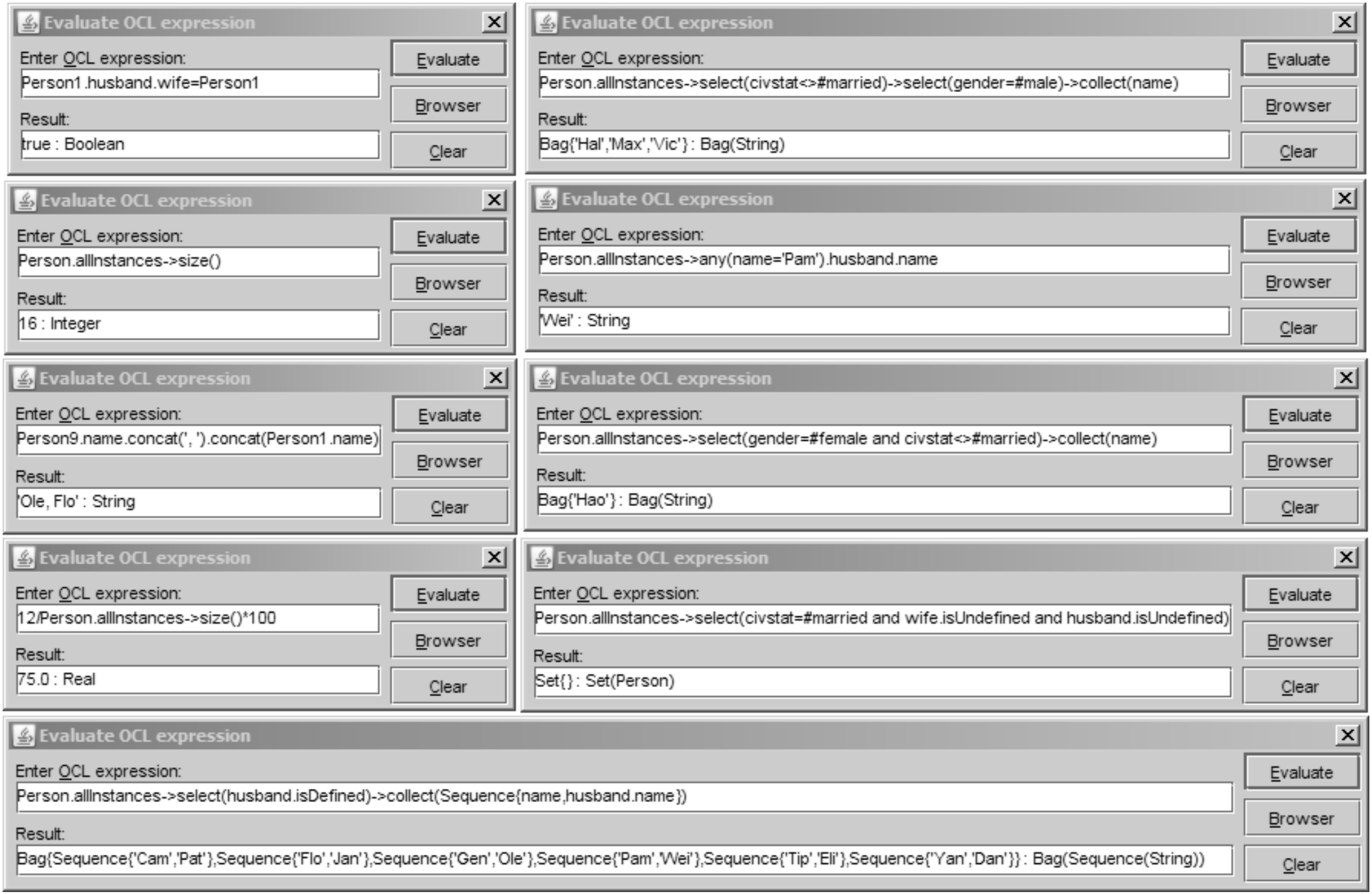

Log
checking structure...
Multiplicity constraint violation in association `Marriage':
Object `Person2' of class `Person' is connected to 2 objects of class `Person'
but the multiplicity is specified as `0..1'.
checking structure, found errors.
Ready.


USE: c:/mg/civstat/scp-sub/civstat.use
File Edit State View Help
OCL
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)
Object diagram
Person1:Person
name='Day'
civstat=#widowed
gender=#male
alive=false
Person2:Person
name='Jen'
civstat=#widowed
gender=#male
alive=true
Person3:Person
name='Ina'
civstat=#married
gender=#male
alive=true
wife
wife
Marriage
Marriage
husband
husband
Person4:Person
name='Jan'
civstat=#divorced
gender=#female
alive=false
Person5:Person
name='Hal'
civstat=#divorced
gender=#female
alive=true
Person6:Person
name='Hal'
civstat=#married
gender=#female
alive=true
Person7:Person
name='Bob'
civstat=#single
gender=#male
alive=true
Class invariants
Invariant
Result
Person::attributesDefined
true
Person::femaleHasNoWife
false
Person::maleHasNoHusband
n/a
Person::nameCapitalThenSmallLetters
true
Person::nameIsUnique
false
2 constraints failed.
100%
Evaluation browser
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
Close
Log
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.
Ready.


Object diagram
Person3:Person
name='Hao'
civstat=#single
gender=#female
alive=true
Person1:Person
name='Flo'
civstat=#married
gender=#female
alive=true
Person2:Person
name='Cam'
civstat=#married
gender=#female
alive=true
Person4:Person
name='Yan'
civstat=#married
gender=#female
alive=true
Person5:Person
name='Gen'
civstat=#married
gender=#female
alive=true
Person6:Person
name='Tip'
civstat=#married
gender=#female
alive=true
Person7:Person
name='Pam'
civstat=#married
gender=#female
alive=true
wife
wife
wife
wife
wife
wife
Marriage
Marriage
Marriage
Marriage
Marriage
Marriage
husband
husband
husband
husband
husband
husband
Person8:Person
name='Pat'
civstat=#married
gender=#male
alive=true
Person9:Person
name='Ole'
civstat=#married
gender=#male
alive=true
Person10:Person
name='Jan'
civstat=#married
gender=#male
alive=true
Person12:Person
name='Wei'
civstat=#married
gender=#male
alive=true
Person13:Person
name='Dan'
civstat=#married
gender=#male
alive=true
Person15:Person
name='Eli'
civstat=#married
gender=#male
alive=true
Person11:Person
name='Max'
civstat=#single
gender=#male
alive=true
Person14:Person
name='Hal'
civstat=#single
gender=#male
alive=true
Person16:Person
name='Vic'
civstat=#single
gender=#male
alive=true

```
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;
```

```
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 ...
     !set ...
     !insert ...

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.
```

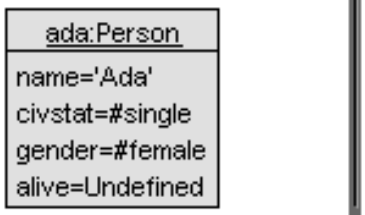

Evaluate OCL expression
Enter OCL expression:
Person1.husband.wife=Person1
Result:
true : Boolean
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(civstat<>#married)->select(gender=#male)->collect(name)
Result:
Bag{'Hal','Max','Vic'} : Bag(String)
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->size()
Result:
16 : Integer
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->any(name='Pam').husband.name
Result:
'Wei' : String
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person9.name.concat(', ').concat(Person1.name)
Result:
'Ole, Flo' : String
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(gender=#female and civstat<>#married)->collect(name)
Result:
Bag{'Hao'} : Bag(String)
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
12/Person.allInstances->size()*100
Result:
75.0 : Real
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(civstat=#married and wife.isUndefined and husband.isUndefined)
Result:
Set{} : Set(Person)
Evaluate
Browser
Clear
Evaluate OCL expression
Enter OCL expression:
Person.allInstances->select(husband.isDefined)->collect(Sequence{name,husband.name})
Result:
Bag{Sequence{'Cam','Pat'},Sequence{'Flo','Jan'},Sequence{'Gen','Ole'},Sequence{'Pam','Wei'},Sequence{'Tip','Eli'},Sequence{'Yan','Dan'}} : Bag(Sequence(String))
Evaluate
Browser
Clear

# Independence of invariants

**Class extent**

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

**Object diagram**



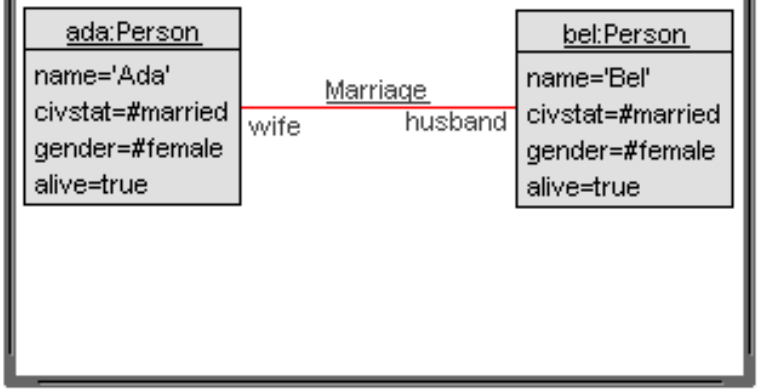

**Command list**

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)

**Class extent**

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

**Object diagram**



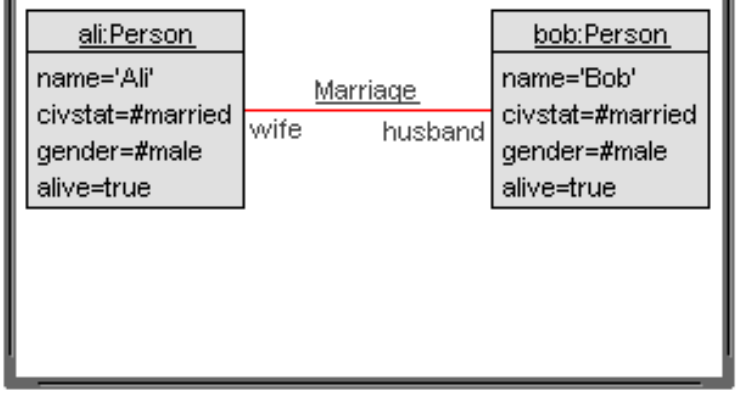

**Command list**

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

**Class extent**

| Person | name | civstat | gender | alive | attributesDefined | femaleHasNoWife | maleHasNoHusband | nameCapitalThenSmallLetters | nameIsUnique |
|---|---|---|---|---|---|---|---|---|---|
| ali | 'Ali' | #married | #male | true | ✔ | ✔ | ✘ | ✔ | ✔ |
| bob | 'Bob' | #married | #male | true | ✔ | ✔ | ✔ | ✔ | ✔ |

**Object diagram**



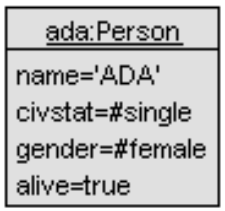

**Command list**

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

**Class extent**

| Person | name | civstat | gender | alive | attributesDefined | femaleHasNoWife | maleHasNoHusband | nameCapitalThenSmallLetters | nameIsUnique |
|---|---|---|---|---|---|---|---|---|---|
| ada | 'ADA' | #single | #female | true | ✔ | ✔ | ✔ | ✘ | ✔ |

**Object diagram**



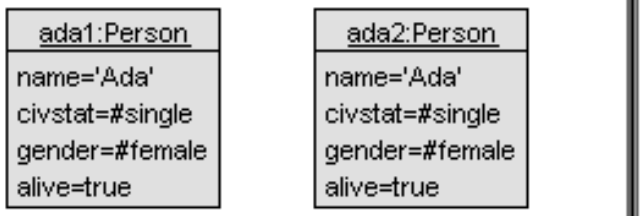

**Command list**

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

**Class extent**

| Person | name | civstat | gender | alive | attributesDefined | femaleHasNoWife | maleHasNoHusband | nameCapitalThenSmallLetters | nameIsUnique |
|---|---|---|---|---|---|---|---|---|---|
| ada1 | 'Ada' | #single | #female | true | ✔ | ✔ | ✔ | ✔ | ✘ |
| ada2 | 'Ada' | #single | #female | true | ✔ | ✔ | ✔ | ✔ | ✘ |

**Object diagram**

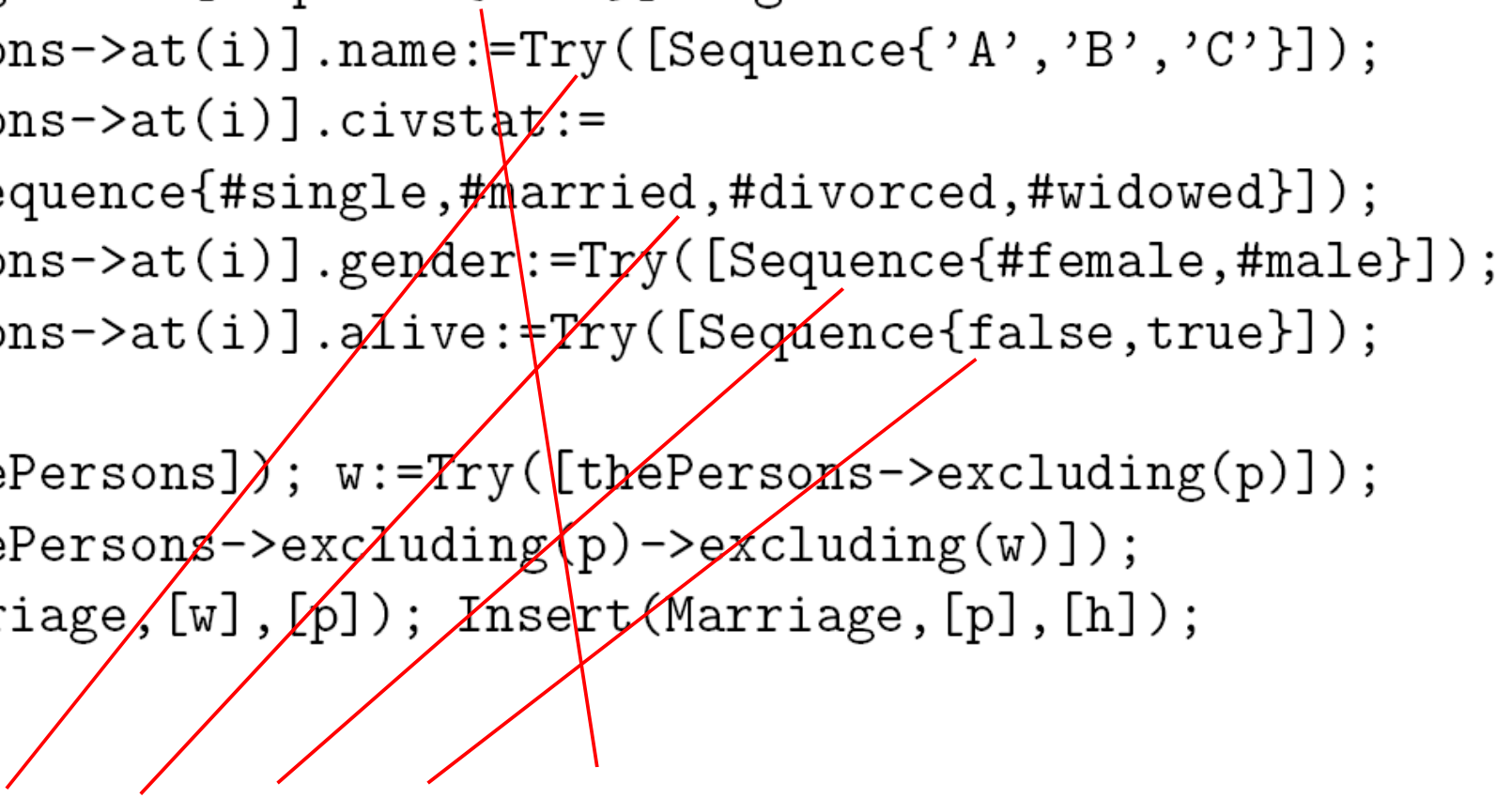



**Command list**

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

```
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.

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

```
procedure attemptBigamy()
var p: Person, w: Person, h:Person, thePersons: Sequence(Person);
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;
```



# Try VERSUS Any

- threeWomenTryPlain()
- threeWomenTryReject()
- threeWomenAny()

| | RandomStart | StateNumber | In-/Valid | Order |
|---|---|---|---|---|
| 1. Call threeWomenTryPlain() | 937 | 6 | Valid | ABC |
| 2. Call threeWomenTryPlain() | 2543 | 6 | Valid | ABC |
| 3. Call threeWomenTryReject() | 593 | 1 | Valid | ABC |
| 4. Call threeWomenTryReject() | 8254 | 1 | Valid | ABC |
| 5. Call threeWomenAny() | 3177 | 1 | Invalid | BCC |
| 6. Call threeWomenAny() | 2932 | 1 | Valid | CBA |

- threeWomenTryPlain()  always yields a valid state with 6 states checked
- threeWomenTryReject() always yields a valid state with 1 state  checked
- threeWomenAny()
  valid   state probalility : 3*2*1/3*3*3 = 6/27 =~= 0.22
  invalid state probalility : 1-(6/27)           =~= 0.78

```
procedure threeWomenTryPlain()
var theFemales: Sequence(Person);
begin
theFemales:=CreateN(Person,[3]);
for i:Integer in [Sequence{1..3}]
  begin [theFemales->at(i)].name:=Try([Sequence{'Ada','Bel','Cam'}]);
  [theFemales->at(i)].civstat:=[#single];
  [theFemales->at(i)].gender:=[#female];
  [theFemales->at(i)].alive:=[true]; end;
end;

procedure threeWomenTryReject()
var theFemales: Sequence(Person);
begin
theFemales:=CreateN(Person,[3]);
for i:Integer in [Sequence{1..3}]
  begin [theFemales->at(i)].name:=Try([Sequence{'Ada','Bel','Cam'}
    ->reject(n|Person.allInstances->exists(p|p.name=n))]);
  ... end;
end;

procedure threeWomenAny()
var theFemales: Sequence(Person);
begin
theFemales:=CreateN(Person,[3]);
for i:Integer in [Sequence{1..3}]
  begin [theFemales->at(i)].name:=Any([Sequence{'Ada','Bel','Cam'}]);
  ... end;
end;
```

```
use> open civstat.use

use> gen start civstat.assl threeWomenTryPlain()
use> gen result
     Random number generator was initialized with 937.
     Checked 6 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Ada'
     !set @Person1.civstat := #single
     !set @Person1.gender := #female
     !set @Person1.alive := true
     !set @Person2.name := 'Bel'
     !set @Person2.civstat := #single
     !set @Person2.gender := #female
     !set @Person2.alive := true
     !set @Person3.name := 'Cam'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
use> gen result accept
     Generated result (system state) accepted.

-- 1. 'Ada' 'Ada' 'Ada'
-- 2. 'Ada' 'Ada' 'Bel'
-- 3. 'Ada' 'Ada' 'Cam'
-- 4. 'Ada' 'Bel' 'Ada'
-- 5. 'Ada' 'Bel' 'Bel'
-- 6. 'Ada' 'Bel' 'Cam'
```

```
use> reset

use> gen start civstat.assl threeWomenTryPlain()
use> gen result
     Random number generator was initialized with 2543.
     Checked 6 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Ada'
     !set @Person1.civstat := #single
     !set @Person1.gender := #female
     !set @Person1.alive := true
     !set @Person2.name := 'Bel'
     !set @Person2.civstat := #single
     !set @Person2.gender := #female
     !set @Person2.alive := true
     !set @Person3.name := 'Cam'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
use> gen result accept
     Generated result (system state) accepted.
```

```
use> reset

use> gen start civstat.assl threeWomenTryReject()
use> gen result
     Random number generator was initialized with 593.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Ada'
     !set @Person1.civstat := #single
     !set @Person1.gender := #female
     !set @Person1.alive := true
     !set @Person2.name := 'Bel'
     !set @Person2.civstat := #single
     !set @Person2.gender := #female
     !set @Person2.alive := true
     !set @Person3.name := 'Cam'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
use> gen result accept
     Generated result (system state) accepted.

-- 1. 'Ada' 'Bel' 'Cam'
-- 2. 'Bel' 'Cam'
-- 3. 'Cam'
```

```
use> reset

use> gen start civstat.assl threeWomenTryReject()
use> gen result
     Random number generator was initialized with 8254.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Ada'
     !set @Person1.civstat := #single
     !set @Person1.gender := #female
     !set @Person1.alive := true
     !set @Person2.name := 'Bel'
     !set @Person2.civstat := #single
     !set @Person2.gender := #female
     !set @Person2.alive := true
     !set @Person3.name := 'Cam'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
use> gen result accept
     Generated result (system state) accepted.
```

```
use> reset

use> gen start civstat.assl threeWomenAny()

use> gen result
     Random number generator was initialized with 3177.
     Checked 1 snapshots.
     Result: No valid state found.
use> gen result accept
     No commands available.
```

```
use> reset

use> gen start civstat.assl threeWomenAny()

use> gen result
     Random number generator was initialized with 2932.
     Checked 1 snapshots.
     Result: Valid state found.
     Commands to produce the valid state:
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Cam'
     !set @Person1.civstat := #single
     !set @Person1.gender := #female
     !set @Person1.alive := true
     !set @Person2.name := 'Bel'
     !set @Person2.civstat := #single
     !set @Person2.gender := #female
     !set @Person2.alive := true
     !set @Person3.name := 'Ada'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
use> gen result accept
     Generated result (system state) accepted.
```

```
use> reset

-- -r <number> start for random number generator
-- -b print brief information about state changes

use> gen start -r 3177 -b civstat.assl threeWomenAny()
     !create Person1,Person2,Person3 : Person
     !set @Person1.name := 'Bel'
     !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 := 'Cam'
     !set @Person3.civstat := #single
     !set @Person3.gender := #female
     !set @Person3.alive := true
     check state (1): Person::nameIsUnique  invalid.
     undo: !set @Person3.alive := true
     ...
     undo: !create Person1,Person2,Person3 : Person
use> gen result
     Random number generator was initialized with 3177.
     Checked 1 snapshots.
     Result: No valid state found.
use> gen result accept
     No commands available.
```

Thanks for your attention!