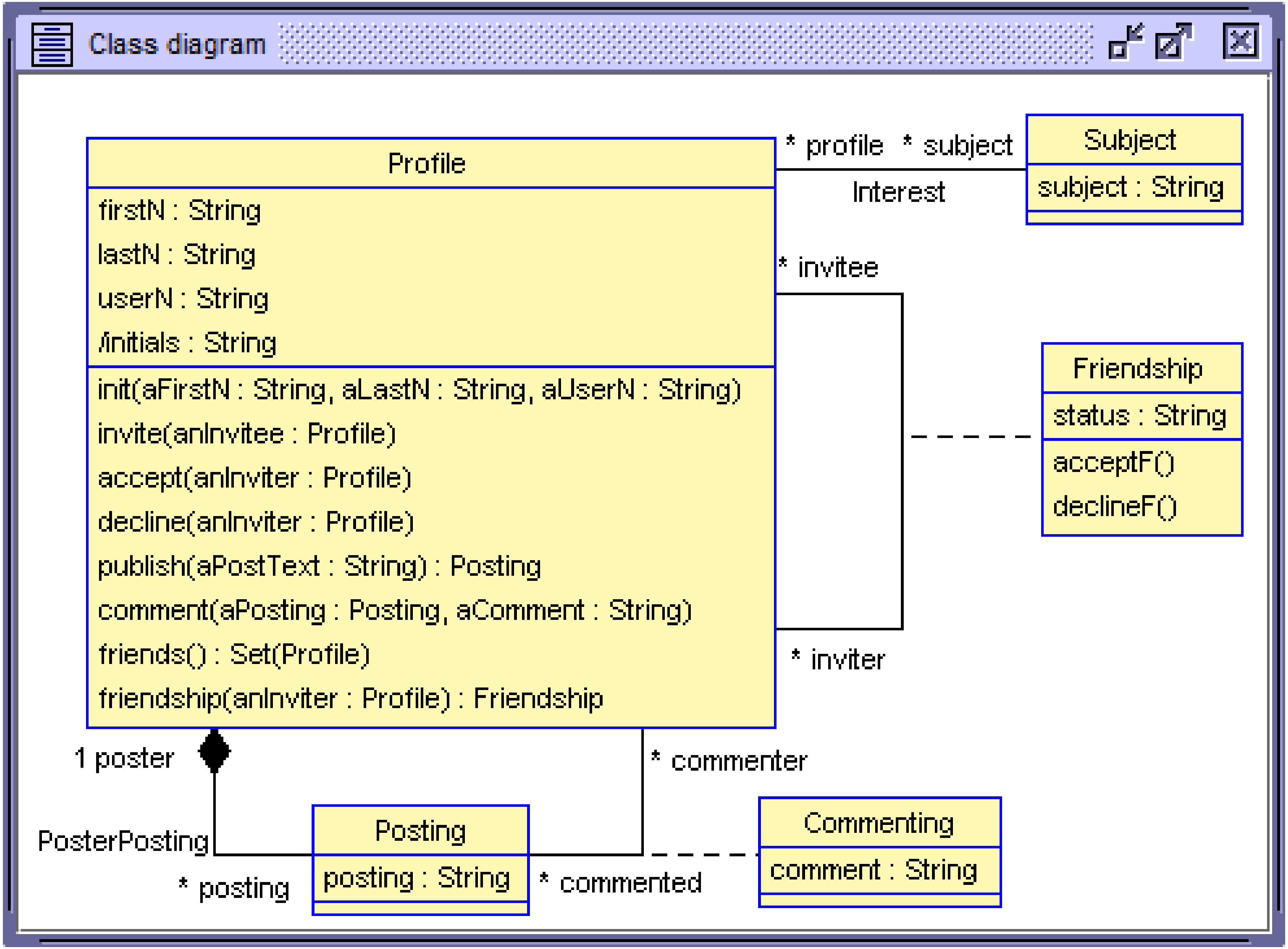
Design of Information Systems

OCL Collection Concepts and Collection Operations

Martin Gogolla
University of Bremen, Germany
Database Systems Group

Collections

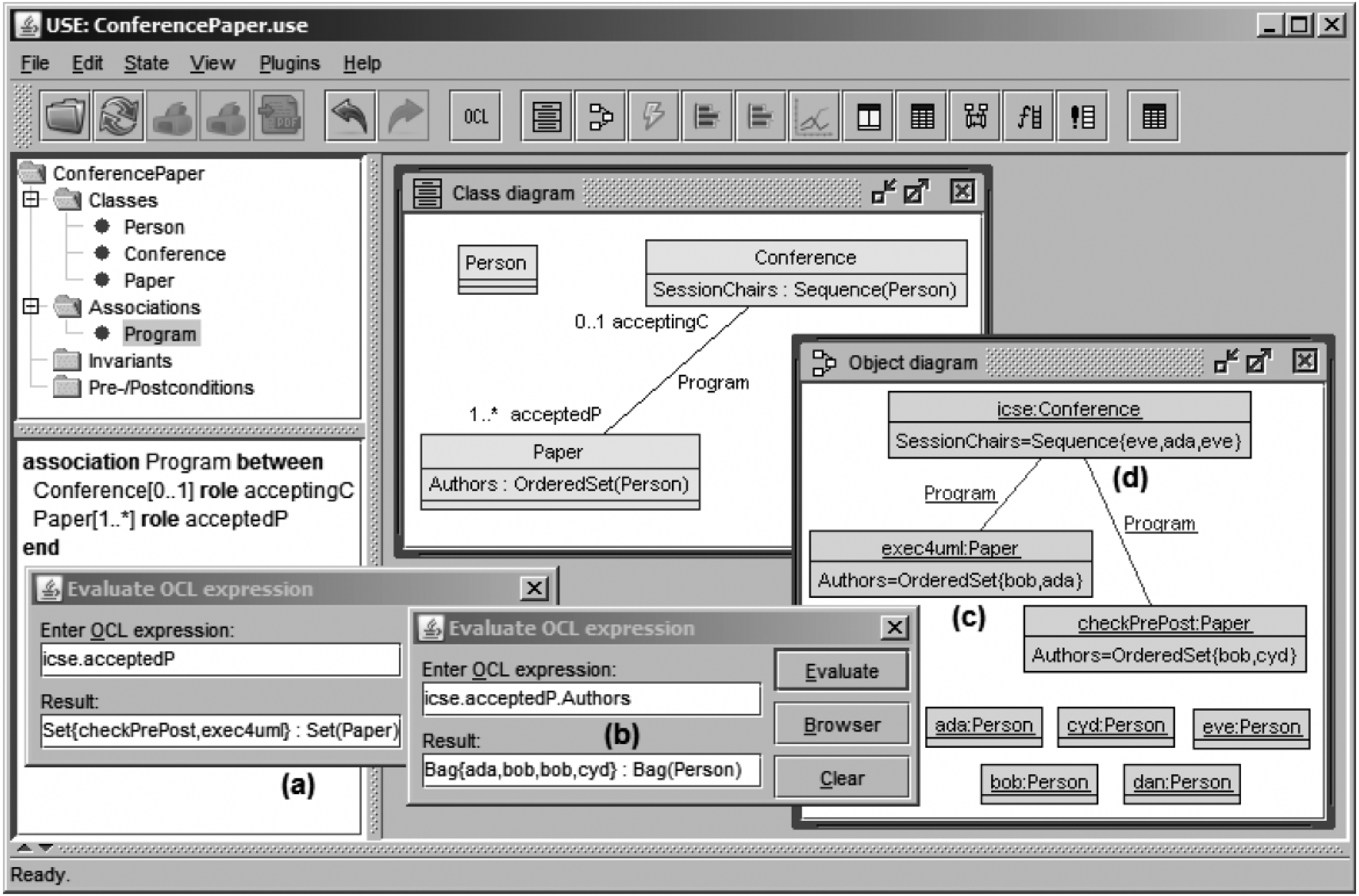
- Common in modeling and programming languages
- "*A collection (or container) is a grouping of some variable number of data items (possibly zero) that ... need to be operated upon together in some controlled fashion.*" Wikipedia
- Examples: set, list, multi-set (allowing duplicates), stack, ...
- UML collections: Set, Bag, Sequence, OrderedSet, Tuple
- Parametrized with element type(s) and access option (for Tuple)


Class diagram
Profile
firstN : String
lastN : String
userN : String
/initials : String
init(aFirstN : String, aLastN : String, aUserN : String)
invite(anInvitee : Profile)
accept(anInviter : Profile)
decline(anInviter : Profile)
publish(aPostText : String) : Posting
comment(aPosting : Posting, aComment : String)
friends() : Set(Profile)
friendship(anInviter : Profile) : Friendship
* profile
* subject
Interest
Subject
subject : String
* invitee
Friendship
status : String
acceptF()
declineF()
* inviter
1 poster
PosterPosting
* posting
Posting
posting : String
* commenter
* commented
Commenting
comment : String

Example collections in SocialNetwork

```
merkel.inviter: Set(Profile)

merkel.posting: Set(Posting)

merkel.posting.commenter: Bag(Profile)

-- !create merkel,putin,trump:Profile
Sequence{merkel,putin,trump}: Sequence(Profile)

OrderedSet{merkel,putin,trump}: OrderedSet(Profile)

Sequence{merkel,putin,trump,may}.yearE = Sequence{2005,2000,2016,2016}
-- yearE: year of first election; imaginable for example model

OrderedSet{2005,2000,2016,2016} = OrderedSet{2005,2000,2016}

-- Paper::authors:OrderedSet(Author); more precise than Sequence(Author)

Sequence{may,merkel}->collect(p|Tuple{L:p.lastN,I:p.initials}) =
  Sequence{Tuple{L='May',   I='TM'},
           Tuple{L='Merkel',I='AM'}}: Sequence(Tuple(L:String,I:String))
```

Example collections in ConferenceWorld



```
association Program between
  Conference[0..1] role acceptingC
  Paper[1..*] role acceptedP
end
```

Evaluate OCL expression

Enter OCL expression: icse.acceptedP

Result: Set{checkPrePost,exec4uml} : Set(Paper)

Evaluate OCL expression

Enter OCL expression: icse.acceptedP.Authors

Result: Bag{ada,bob,bob,cyd} : Bag(Person)

Collection parameters and collection syntax

- Type kinds with type paramenters: Set(T), Bag(T), Sequence(T), OrderedSet(T), Tuple(A1:T1,...,An:Tn); access Ai
- Abstract type kind (no instances) Collection(T) generalization of Set(T), Bag(T), Sequence(T), OrderedSet(T)
- Parameter actualization in order to build types
- Types always written with parentheses ( )

```
Set(Posting), Bag(Profile),
Sequence(Profile), OrderedSet(Integer),
Tuple(L:String,I:String)
```

- Instantiations always written with braces { }

```
Set{merkel,trump}, Bag{trump,putin,trump},
Sequence{merkel,putin,trump}, OrderedSet{2005,2000,2016},
Tuple{L='Merkel',I='AM'}
```

- Tuple access `Tuple{L='Merkel',I='AM'}.I='AM'`

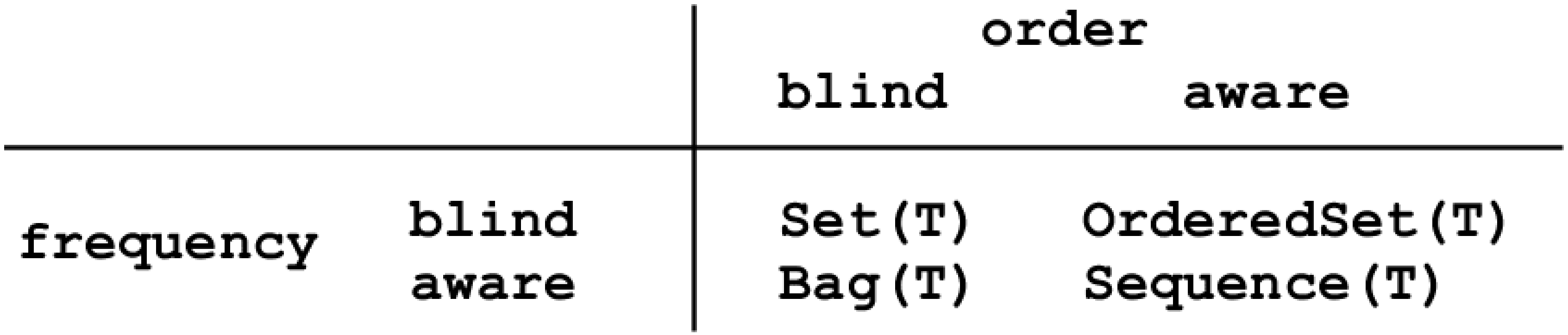
Collection properties (for homogeneous collections)

- Two criteria in order to distinguish between collections:
  Insertion order and insertion frequency
- Is the insertion order relevant for distinguishing collections?

```
COL->including(E1)->including(E2) = COL->including(E2)->including(E1)
```

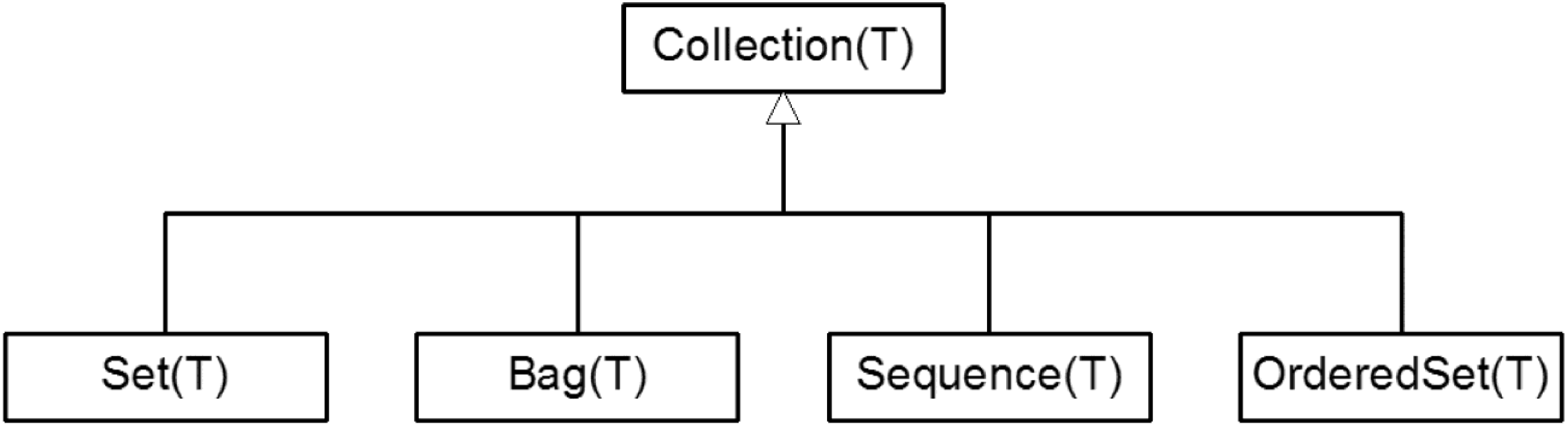
  if required, collection is called order-blind, else order-aware
- Is the insertion frequency relevant for distinguishing collections?

```
COL->includes(E) implies COL->including(E) = COL
```

  if required, collection is called frequency-blind, else frequency-aware
- 

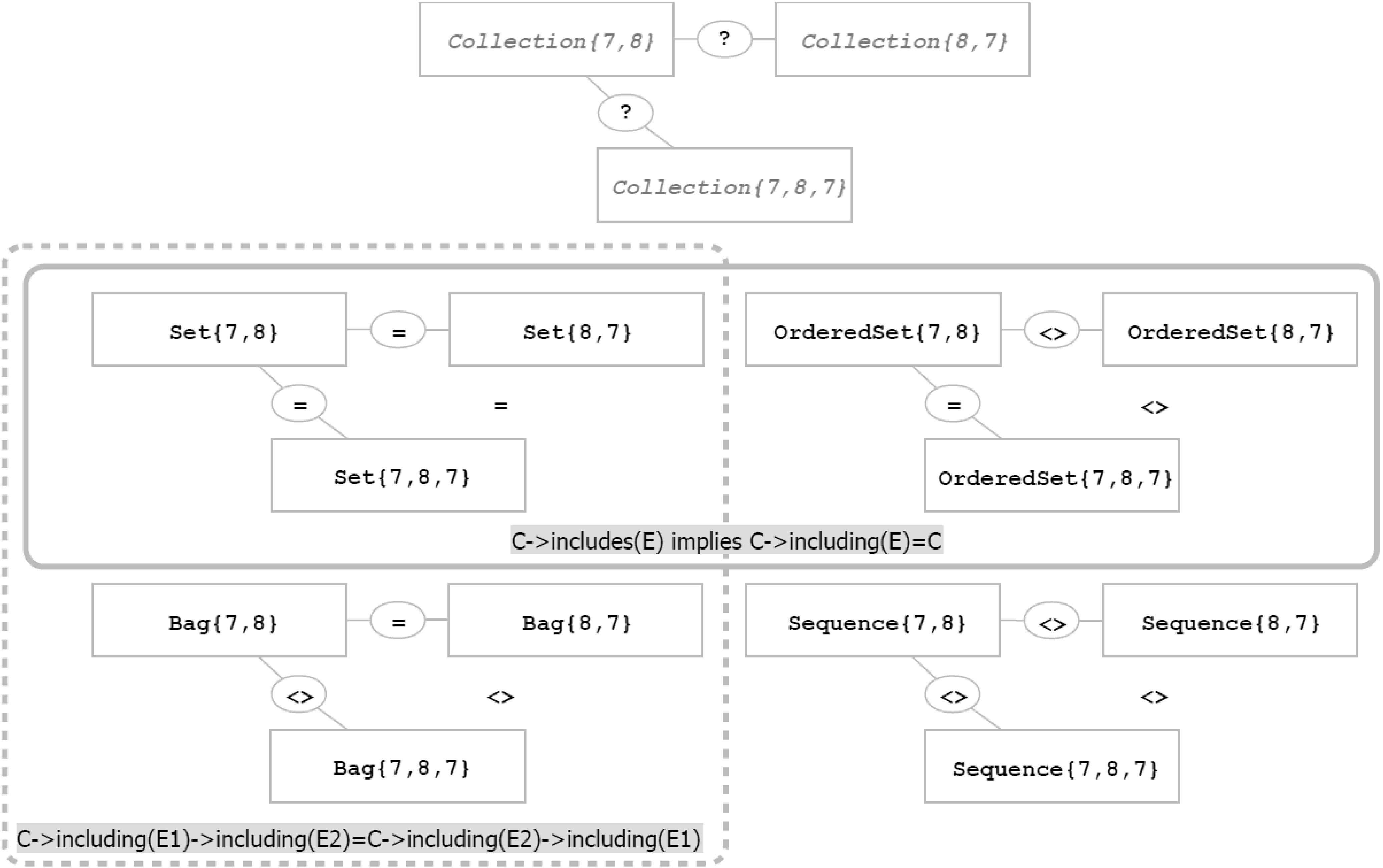
		order	
		blind	aware
frequency	blind	Set(T)	OrderedSet(T)
	aware	Bag(T)	Sequence(T)

Collection type hierarchy and properties



- order-blind and frequency-blind Set(T)
- order-blind and frequency-aware Bag(T)
- order-aware and frequency-aware Sequence(T)
- order-aware and frequency-blind OrderedSet(T)
- OCL 1.3 only had Set(T), Bag(T), Sequence(T)
- OCL 1.4 added OrderedSet(T)
- also used: order-insensible/-sensible, frequency-insensible/-sensible

Collection properties: Insertion order and frequency



Collection{7,8} ? *Collection{8,7}*

?

Collection{7,8,7}

Set{7,8} = Set{8,7}

= =

Set{7,8,7}

OrderedSet{7,8} <> OrderedSet{8,7}

= <>

OrderedSet{7,8,7}

C->includes(E) implies C->including(E)=C

Bag{7,8} = Bag{8,7}

<> <>

Bag{7,8,7}

Sequence{7,8} <> Sequence{8,7}

<> <>

Sequence{7,8,7}

C->including(E1)->including(E2)=C->including(E2)->including(E1)

Collection properties

```
use> !C:=Set{Set{7,8}, Set{8,7},
             Set{7,8,8}, Set{8,7,7}}
use> ?C
     Set{Set{7,8}} : Set(Set(Integer))

use> !D:=Set{Bag{7,8}, Bag{8,7},
             Bag{7,8,8}, Bag{8,7,7}}
use> ?D
     Set{Bag{7,8}, Bag{7,7,8}, Bag{7,8,8}} : Set(Bag(Integer))

use> !E:=Set{OrderedSet{7,8}, OrderedSet{8,7},
             OrderedSet{7,8,8}, OrderedSet{8,7,7}}
use> ?E
     Set{OrderedSet{7,8}, OrderedSet{8,7}} : Set(OrderedSet(Integer))

use> !F:=Set{Sequence{7,8}, Sequence{8,7},
             Sequence{7,8,8}, Sequence{8,7,7}}
use> ?F
     Set{Sequence{7,8}, Sequence{8,7},
         Sequence{7,8,8}, Sequence{8,7,7}} : Set(Sequence(Integer))

use> ?Sequence{C->size(), D->size(), E->size(), F->size()}
     Sequence{1, 3, 2, 4} : Sequence(Integer)
```

Collection operations on all collection kinds

Constructors and `destructors'

- Set{...}, Bag{...}, Sequence{...}, OrderedSet{...}
- Set{L..H}, Bag{L..H}, Sequence{L..H}, OrderedSet{L..H} -- Low High
- including(...), excluding(...)

Basic boolean and integer query operations

- =, <>
- includes(...), excludes(...), includesAll(...), excludesAll(...)
- isEmpty(), notEmpty(), size(), count(...)

Advanced boolean query operations

- forAll(...), exists(...), one(...)
- isUnique(...)

Advanced collection-valued query operations

- select(...), reject(...)
- any(...)
- union(...)
- collect(...), collectNested(...)
- flatten()
- sortedBy(...)

Complex query operations: iterate(...), closure(...)

Coercions: asSet(), asBag(), asSequence(), asOrderedSet()

Collection operations on special collection kinds

- first(), last(), at(pos), reverse()
  for order-aware, i.e. Sequence(T), OrderedSet(T)

- subSequence(startPos,endPos) on Sequence(T)

- subOrderedSet(startPos,endPos) on OrderedSet(T)

- intersection(...) for order-blind, i.e. Set(T), Bag(T)

- sum(), min(), max() on Collection(Integer), Collection(Real)

- Few further operations (e.g. indexOf): see OCL standard

Not mentioned yet (and to be discussed further down):
collection operations in the context of generalization
(e.g. for Chess example, c:Character and c.oclIsTypeOf(Knight))

Demonstrating OCL expressions without having objects (Part A)

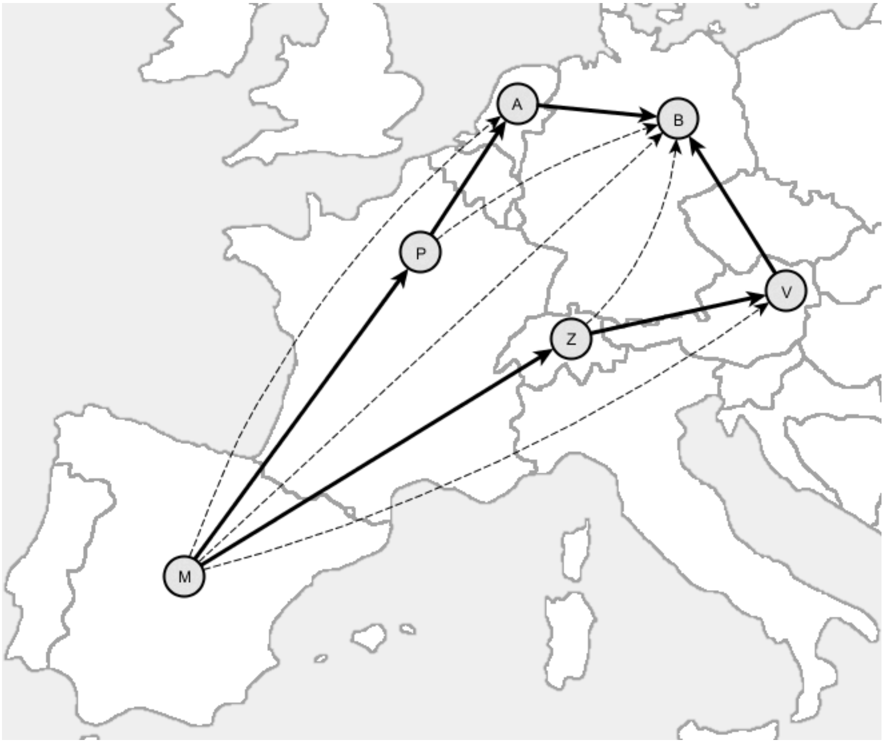
```
Constructors and `destructors'
- Set{7,8}, Bag{7,8,8}, Sequence{7,8,7}, OrderedSet{8,7,7}
- Set{}, Bag{}, Sequence{}, OrderedSet{}
- Set{7..9}, Bag{7..9}, Sequence{7..9}, OrderedSet{7..9}
- Set{}->including(8)->including(7), Bag{8,9,7,8,9}->excluding(9)

Basic boolean and integer query operations
- Set{7,8}=Set{8,7,8,7}, OrderedSet{7,8}<>OrderedSet{8,7}
  Set{7,8}<>Bag{7,8}, OrderedSet{7,8}<>Sequence{8,7}
- Set{7,8}->includes(8), Set{7,8}->excludes(9),
  Set{7,8}->includesAll(Set{8,8,7,7}), Set{7,8}->excludesAll(Set{6,9})
- Set{}->isEmpty(), Set{7,8}->notEmpty(), Set{8,8,7,7}->size()=2
  Set{7,8,7}->count(7), Bag{7,8,7}->count(7)
  Sequence{7,8,7}->count(7), OrderedSet{7,8,7}->count(7)
```

Demonstrating OCL expressions without having objects (Part B)

```
Advanced boolean query operations
- Set{7..9}->forAll(i|i>=0), Bag{7..9}->exists(i|i.mod(2)=0)
- Sequence{7..9}->one(i|i.mod(2)=0)
- OrderedSet{-9..-8}->including(8)->including(9)->isUnique(i|i*i)=false

Advanced collection-valued query operations
- Set{21..42}->select(i|i.mod(3)=0 and i.mod(7)=0)
- Bag{21..42}->reject(i|i.mod(2)=0 or i.mod(3)=0)
- Set{21..42}->any(i|i.mod(2)=1)
- Set{7,8,8}->union(Set{9,9,8}), Bag{7,8,8}->union(Bag{9,9,8})
  Sequence{7,8,8}->union(Sequence{9,9,8})
  OrderedSet{7,8,8}->union(OrderedSet{9,9,8})
- Set{-2..2}->collect(i|i*i), Set{-2..2}->collect(i|Sequence{i,i*i})
  Set{-2..2}->collectNested(i|Sequence{i,i*i})
- Set{-2..2}->collectNested(i|Sequence{i,i*i})->flatten()
- Set{-6,-5,-4,7,8,9}->sortedBy(i|i*i)
```

Demonstrating OCL expressions without having objects (Part C)

```
Complex query operations
- Set{-2..2}->iterate(i:Integer;r:Set(Sequence(OclAny))=Set{}|
    r->including(Sequence{i,i*i,if i.mod(2)=0 then 'E' else 'O' endif}))
- Capitals: M[adrid], P[aris], A[msterdam], B[erlin], Z[urich], V[ienna]
  let TupleSet=
    Set{Tuple{s:'M',t:'P'},Tuple{s:'P',t:'A'},Tuple{s:'A',t:'B'},
        Tuple{s:'M',t:'Z'},Tuple{s:'Z',t:'V'},Tuple{t:'B',s:'V'}} in
  TupleSet->closure(T1|
    TupleSet->select(T2|T1.t=T2.s)->
      collect(T2|Tuple{s:T1.s,t:T2.t})->
        asSet())
                    +-------------+
                    |  select =   |
      Tuple{T1.s,T1.t}   Tuple{T2.s,T2.t}
               |        collect        |
               +-----------------------+
```



Demonstrating OCL expressions without having objects (Part D)

```
Coercions
- Sequence{8,7,8}->asSet()=Set{8,7}
- OrderedSet{8,7,8}->asBag()=Bag{8,7}
- Set{7,8}->asSequence()=Sequence{8,7}
  or Set{7,8}->asSequence()=Sequence{7,8}
- Bag{8,8,7,7}->asOrderedSet()=OrderedSet{7,8}
  or Bag{8,8,7,7}->asOrderedSet()=OrderedSet{8,7}
- Set{-2..2}->collect(i|i*i)->asSet()
```

Thanks for your attention!