OCL Constraints of Analysis Classes

Model driven software development

Dániel Varró

Goals

- How to capture restrictions (constraints) of analysis classes?

- How to capture pre- and postconditions of operations?

What is OCL?

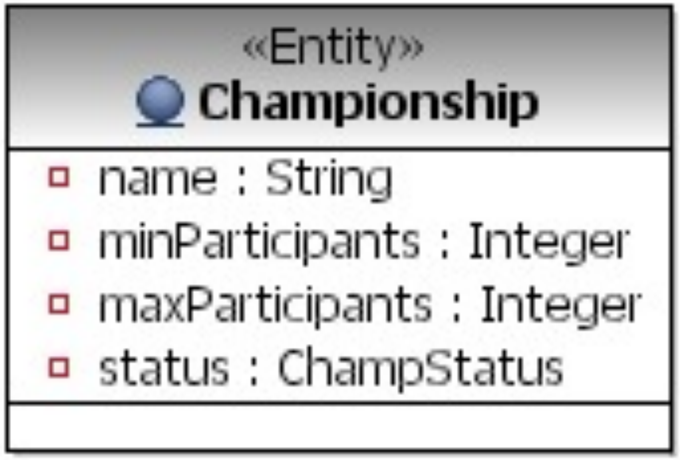
- OCL = Object Constraint Language
- OCL is not a programming language;
  - not possible to write program logic or flow control in OCL
- OCL is a typed language
  - each OCL expression has a type;
  - types within OCL can be any class (kind of Classifier)
- Implementation issues are out of scope and cannot be expressed in OCL

Where to use OCL?

- To specify invariants on classes and types in the class model
- To specify type invariants for Stereotypes
- To describe pre- and postconditions on Operations
- To describe Guards
- As a navigation language
- To specify constraints on operations
- Modeling Language Engineering: well-formedness rules as invariants on the meta-classes in the abstract syntax;

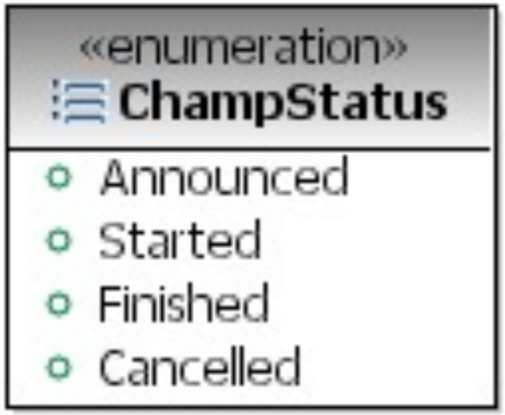
Expressing Invariants on Entity Classes

Informal Constraints on Championship



«Entity»
Championship

- name : String
- minParticipants : Integer
- maxParticipants : Integer
- status : ChampStatus



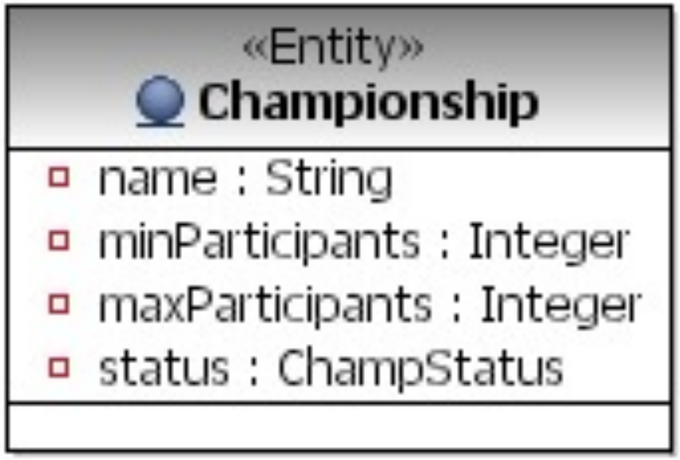
«enumeration»
ChampStatus

- Announced
- Started
- Finished
- Cancelled

What are the restrictions?

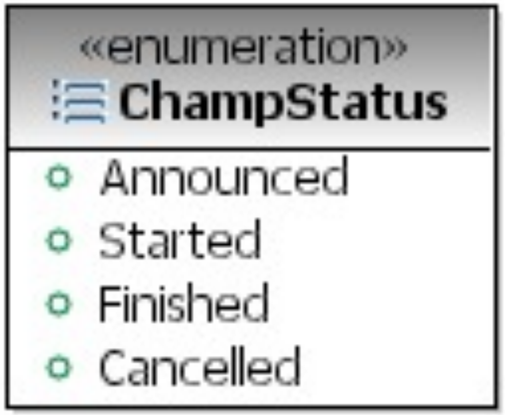
- name is not empty
- minParticipants ≤ maxParticipants
- minParticipants ≥ 0
- maxParticipants > 0

First OCL constraints



«Entity»
Championship

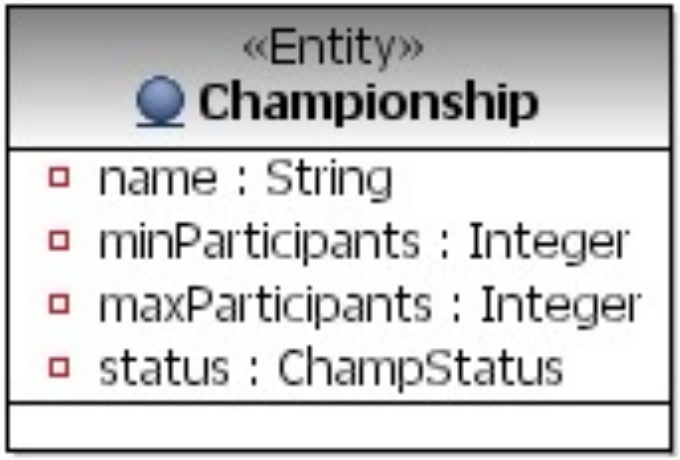
- name : String
- minParticipants : Integer
- maxParticipants : Integer
- status : ChampStatus



«enumeration»
ChampStatus

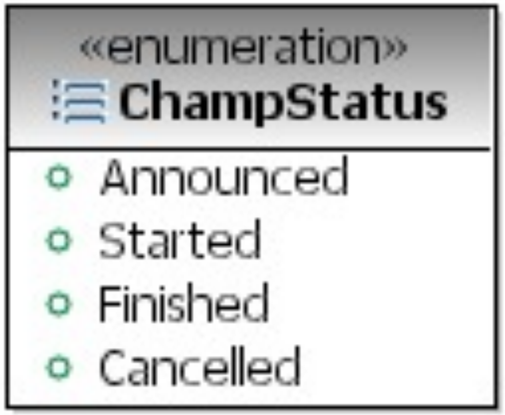
- Announced
- Started
- Finished
- Cancelled

First OCL constraints



«Entity»
Championship

name : String
minParticipants : Integer
maxParticipants : Integer
status : ChampStatus



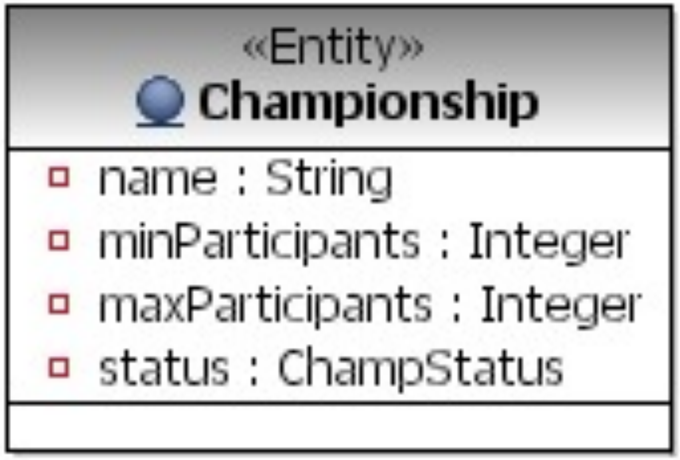
«enumeration»
ChampStatus

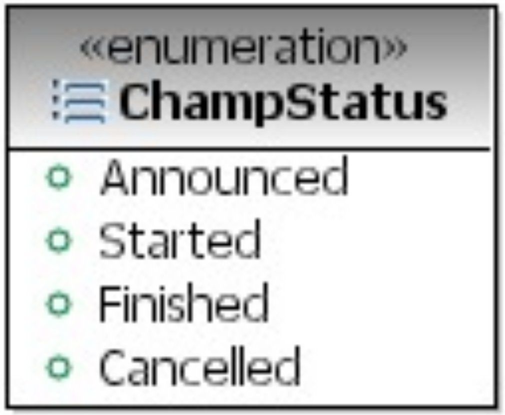
Announced
Started
Finished
Cancelled

- Name is not empty

```
context Championship inv:
   self.name <> ''
```

First OCL constraints



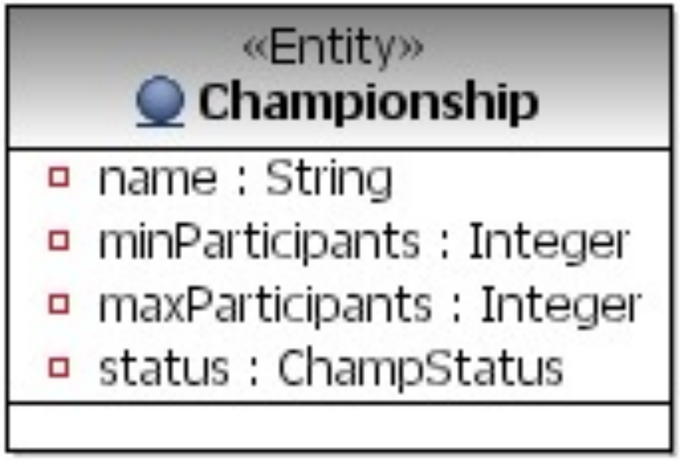


- Name is not empty

```
context Championship inv:
   self.name <> ''
```

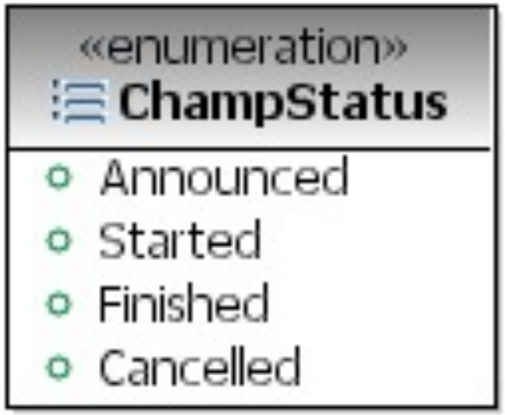
- Constraints on participants

First OCL constraints



«Entity»
Championship

- name : String
- minParticipants : Integer
- maxParticipants : Integer
- status : ChampStatus



«enumeration»
ChampStatus

- Announced
- Started
- Finished
- Cancelled

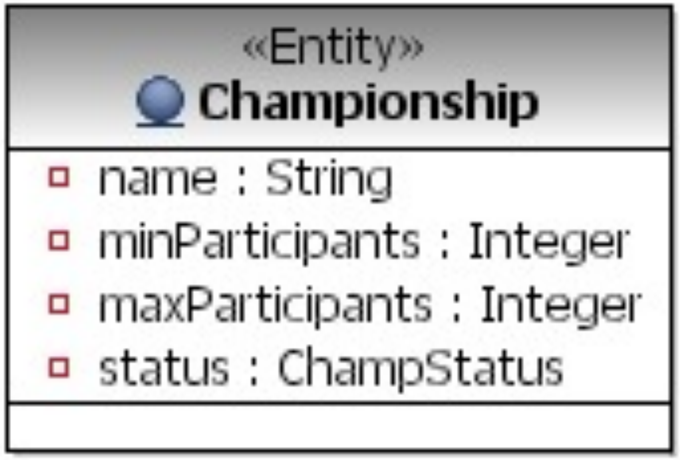
- Name is not empty

```
context Championship inv:
   self.name <> ''
```

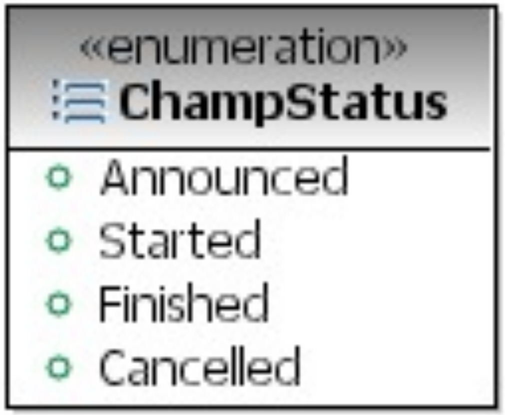
- Constraints on participants

```
context Championship inv:
   self.minParticipants >= 0
```

First OCL constraints



«Entity»
Championship
name : String
minParticipants : Integer
maxParticipants : Integer
status : ChampStatus



«enumeration»
ChampStatus
Announced
Started
Finished
Cancelled

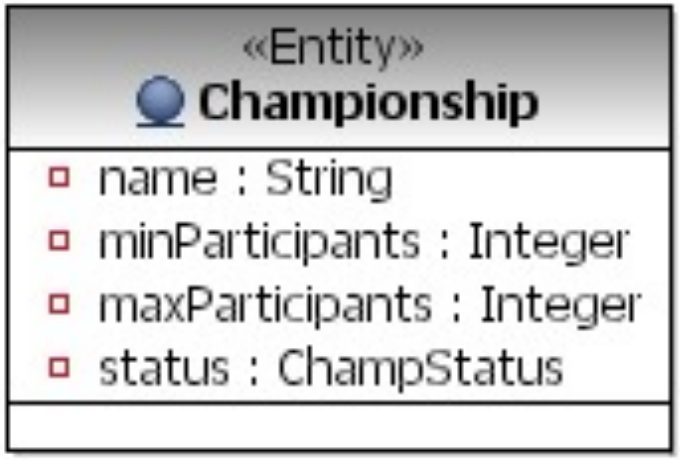
- Name is not empty

```
context Championship inv:
   self.name <> ''
```

- Constraints on participants

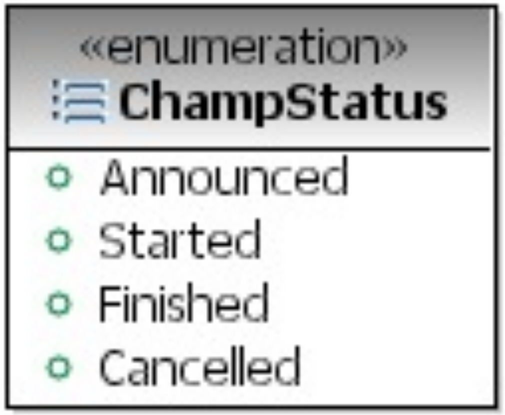
```
context Championship inv:
   self.minParticipants >= 0
context Championship inv:
   self.maxParticipants >= 1
```

First OCL constraints



«Entity»
Championship

- name : String
- minParticipants : Integer
- maxParticipants : Integer
- status : ChampStatus



«enumeration»
ChampStatus

- Announced
- Started
- Finished
- Cancelled

- Name is not empty

```
context Championship inv:
   self.name <> ''
```

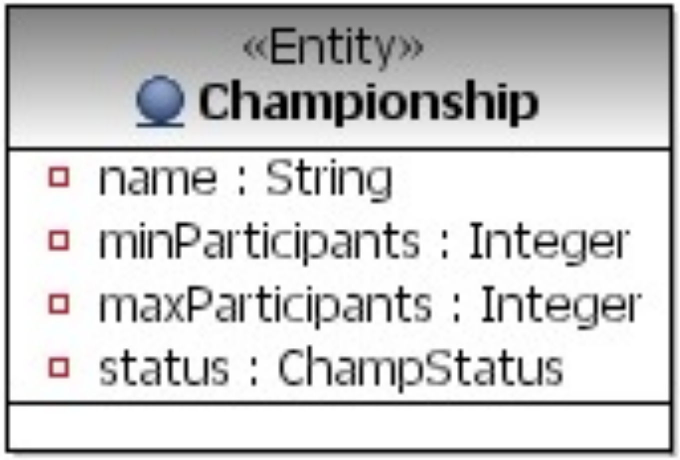
- Constraints on participants

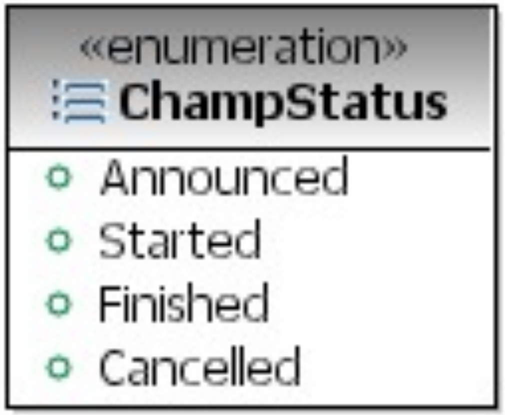
```
context Championship inv:
   self.minParticipants >= 0
```

```
context Championship inv:
   self.maxParticipants >= 1
```

```
context Championship inv:
   self.maxParticipants >=
   self.minParticipants
```

First OCL constraints





Context

Invariant

- Name is not empty

```
context Championship inv:
   self.name <> ''
```

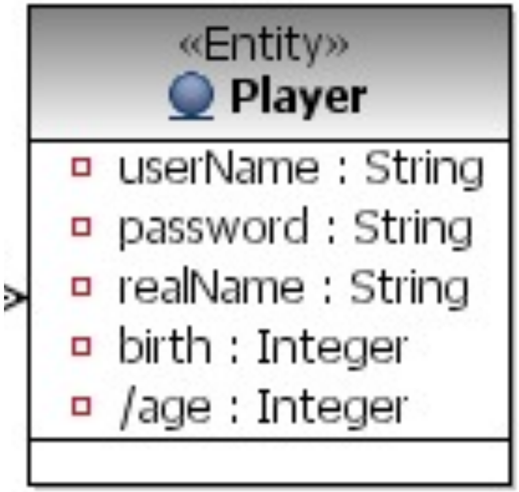
- Constraints on participants

```
context Championship inv:
   self.minParticipants >= 0
context Championship inv:
   self.maxParticipants >= 1
context Championship inv:
   self.maxParticipants >=
   self.m
```

Instance of the class

Navigation along attributes

Informal Constraints on Player

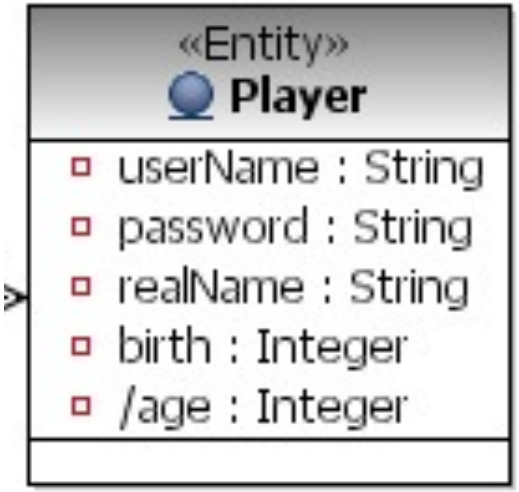


«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- What are the restrictions?
  - userName is not empty
  - userName is unique
  - $1800 \leq birth \leq 3000$
  - password is not empty
  - age = current_year - birth

Informal Constraints on Player

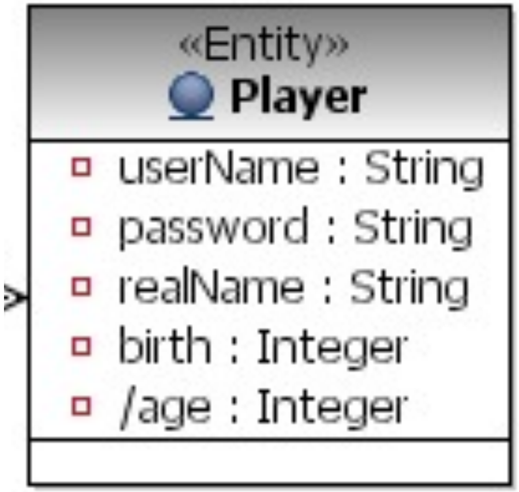


«Entity»

Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

Informal Constraints on Player

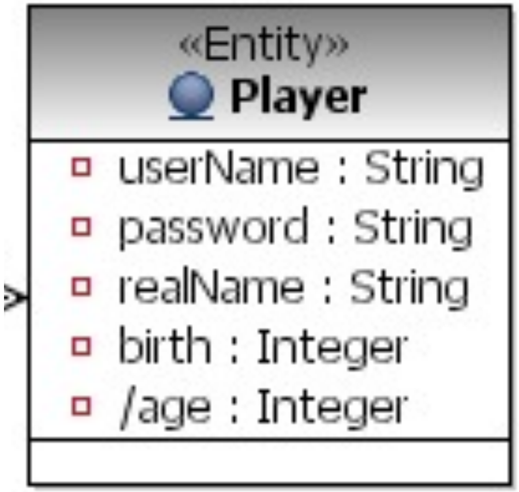


«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- $1800 \leq \text{birth} \leq 3000$

Informal Constraints on Player



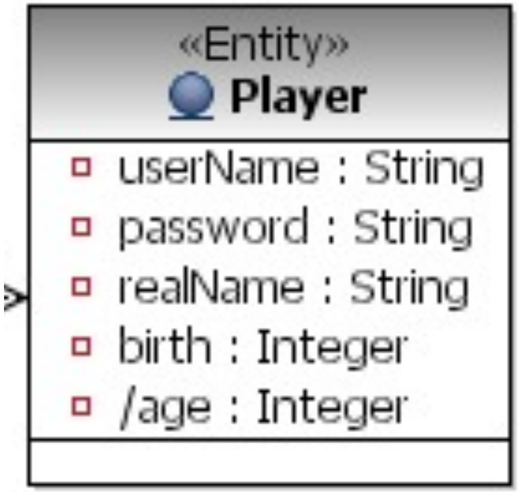
«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Informal Constraints on Player

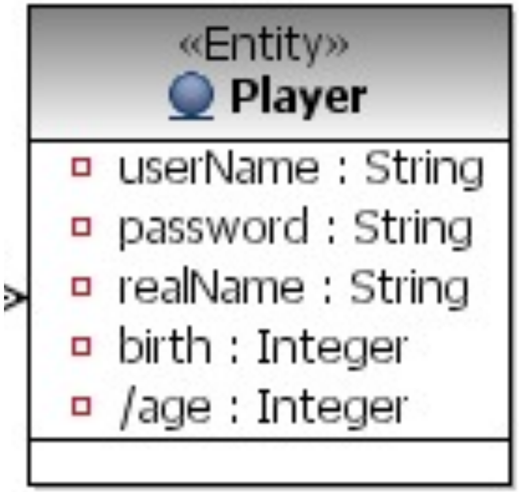


- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

Informal Constraints on Player

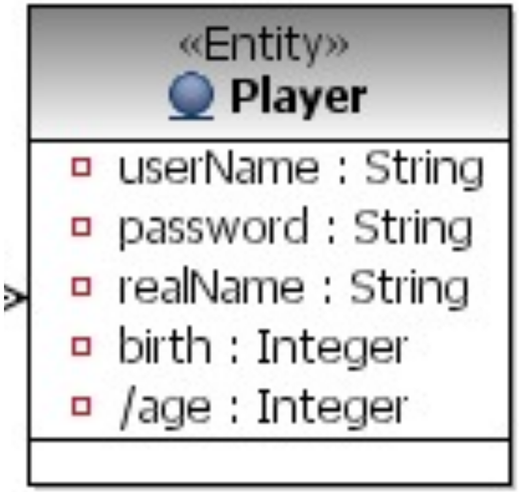

«Entity»
Player
- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

Informal Constraints on Player



«Entity»
Player

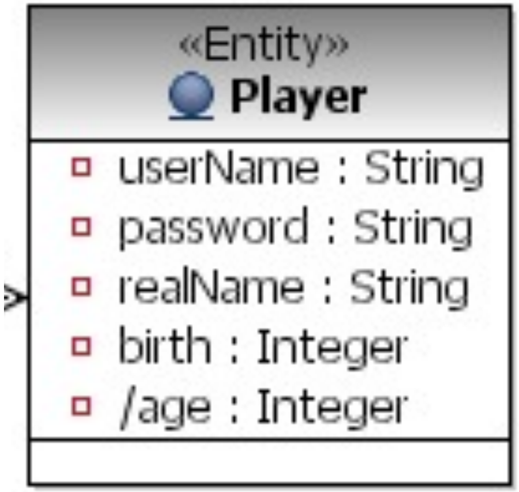
- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- 1800 ≤ birth ≤ 3000

```
context Player inv:
    self.birth >= 1800 and
    self.birth <= 3000
```

Logical AND

Informal Constraints on Player

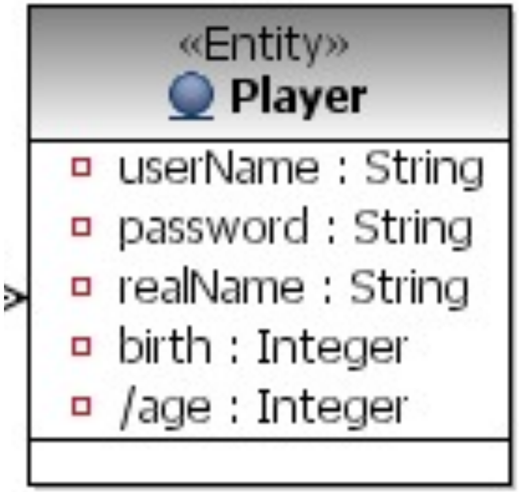


- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

Informal Constraints on Player



«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

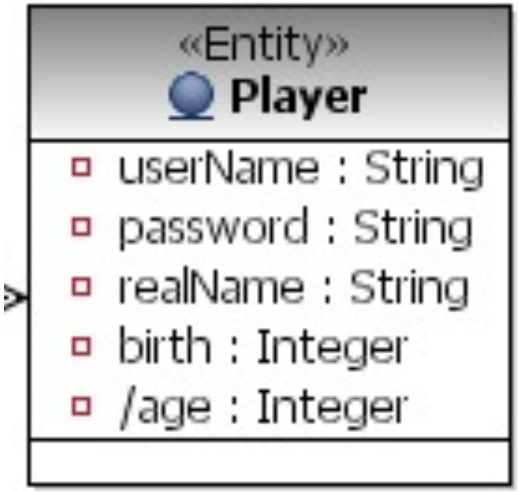
- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

- Name is unique

Informal Constraints on Player



«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- 1800 ≤ birth ≤ 3000

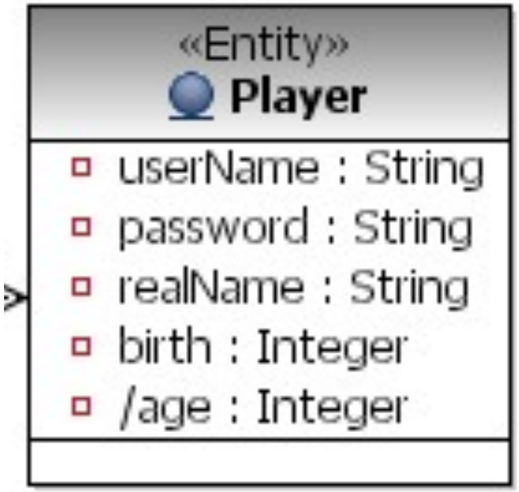
```
context Player inv:
    self.birth >= 1800 and
    self.birth <= 3000
```

Logical AND

- Name is unique

```
context Player inv:
    Player.allInstances->forAll(p1, p2 |
    p1<>p2 implies
    p1.userName <> p2.userName)
```

Informal Constraints on Player



- 1800 ≤ birth ≤ 3000

```
context Player inv:
    self.birth >= 1800 and
    self.birth <= 3000
```

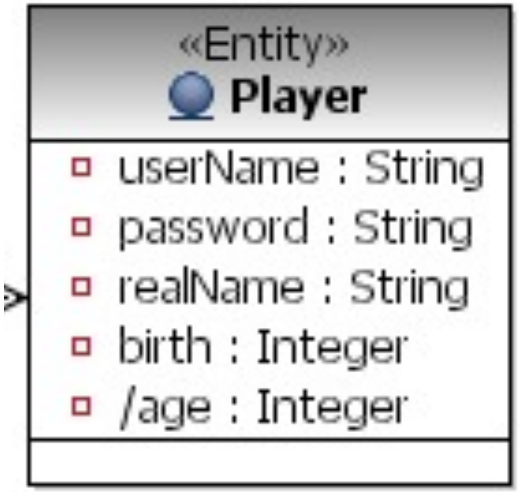
Logical AND

Get all instances into a collection

- Name is unique

```
context Player inv:
    Player.allInstances->forAll(p1, p2 |
    p1<>p2 implies
    p1.userName <> p2.userName)
```

Informal Constraints on Player



- 1800 ≤ birth ≤ 3000

```
context Player inv:
    self.birth >= 1800 and
    self.birth <= 3000
```

Logical AND

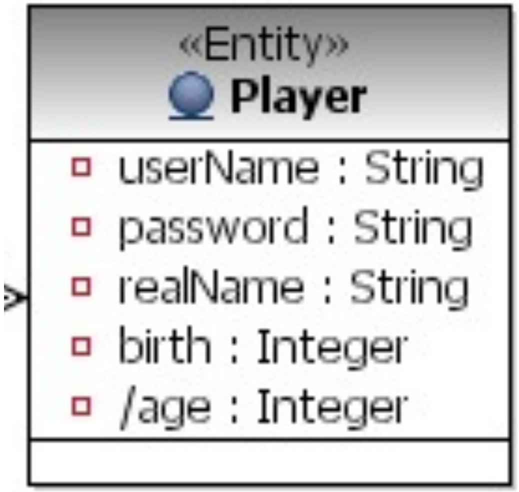
- Name is unique

```
context Player inv:
    Player.allInstances->forAll(p1, p2 |
    p1<>p2 implies
    p1.userName
```

Get all instances into a collection

Universal quantification: For all objects in the collection

Informal Constraints on Player



«Entity»
Player

- userName : String
- password : String
- realName : String
- birth : Integer
- /age : Integer

- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

- Name is unique

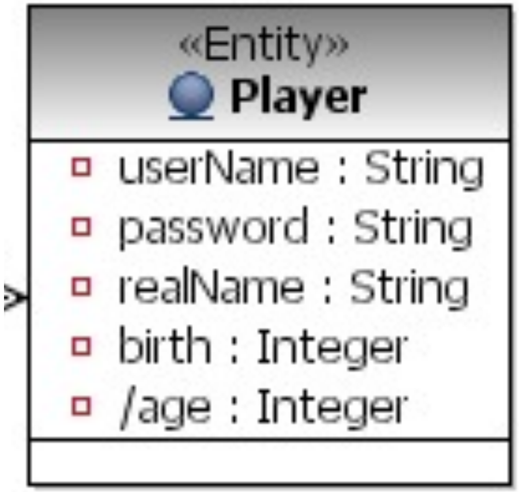
```
context Player inv:
   Player.allInstances->forAll( 1, p2 |
   p1<>p2 implies
   p1.userName
```

Get all instances into a collection

If p1 ≠ p2

Universal quantification: For all objects in the collection

Informal Constraints on Player



- 1800 ≤ birth ≤ 3000

```
context Player inv:
    self.birth >= 1800 and
    self.birth <= 3000
```

Logical AND

Get all instances into a collection

- Name is unique

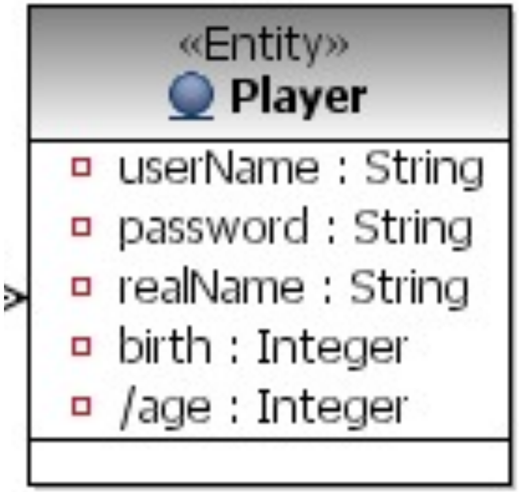
```
context Player inv:
    Player.allInstances->forAll(p1, p2 |
    p1<>p2 implies
    p1.userName
```

If p1 ≠ p2

Then p1.userName ≠ p2.userName

Universal quantification: For all objects in the collection

Informal Constraints on Player


«Entity»
Player
userName : String
password : String
realName : String
birth : Integer
/age : Integer

- 1800 ≤ birth ≤ 3000

```
context Player inv:
   self.birth >= 1800 and
   self.birth <= 3000
```

Logical AND

Get all instances into a collection

Logical implication

- Name is unique

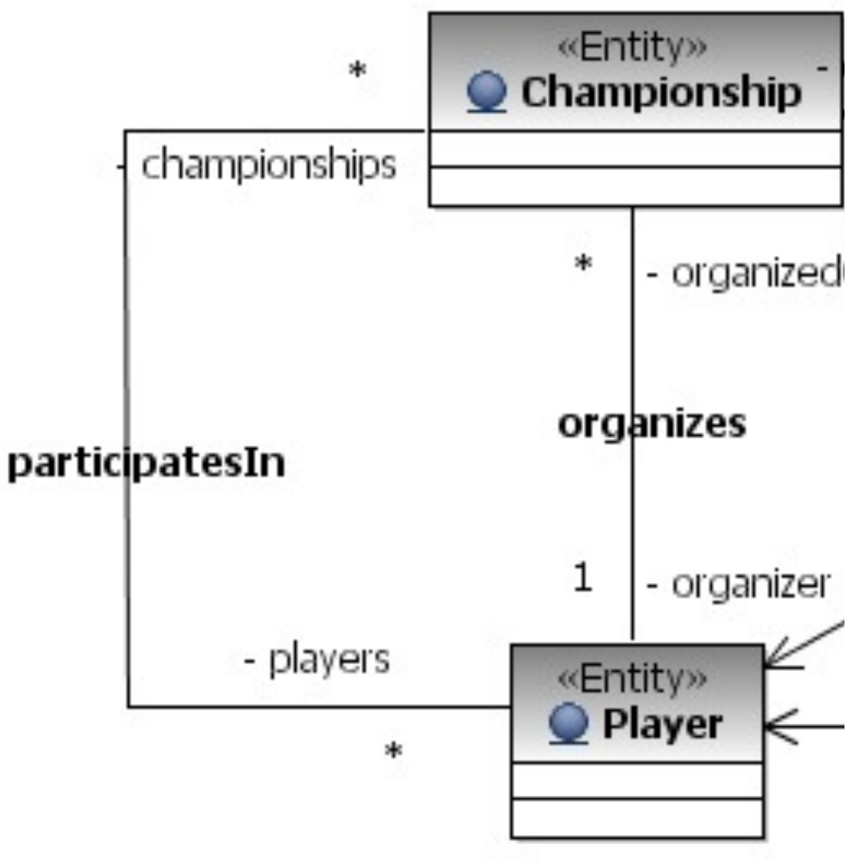
```
context Player inv:
   Player.allInstances->forAll(p1, p2 |
   p1<>p2 implies
   p1.userName
```

If p1 ≠ p2

Then p1.userName ≠ p2.userName

Universal quantification: For all objects in the collection

Properties Automatically Induced by Roles and Multiplicities



organizer:
    Championship -> Player

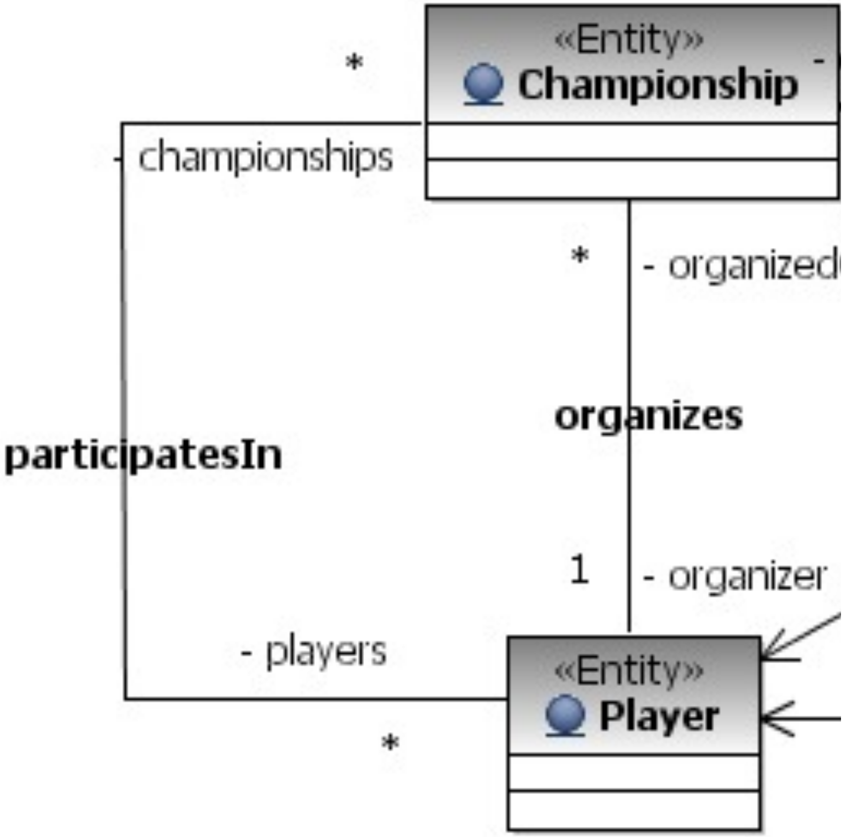
organized:
    Championship -> Set(Player)

championships:
    Championship -> Player

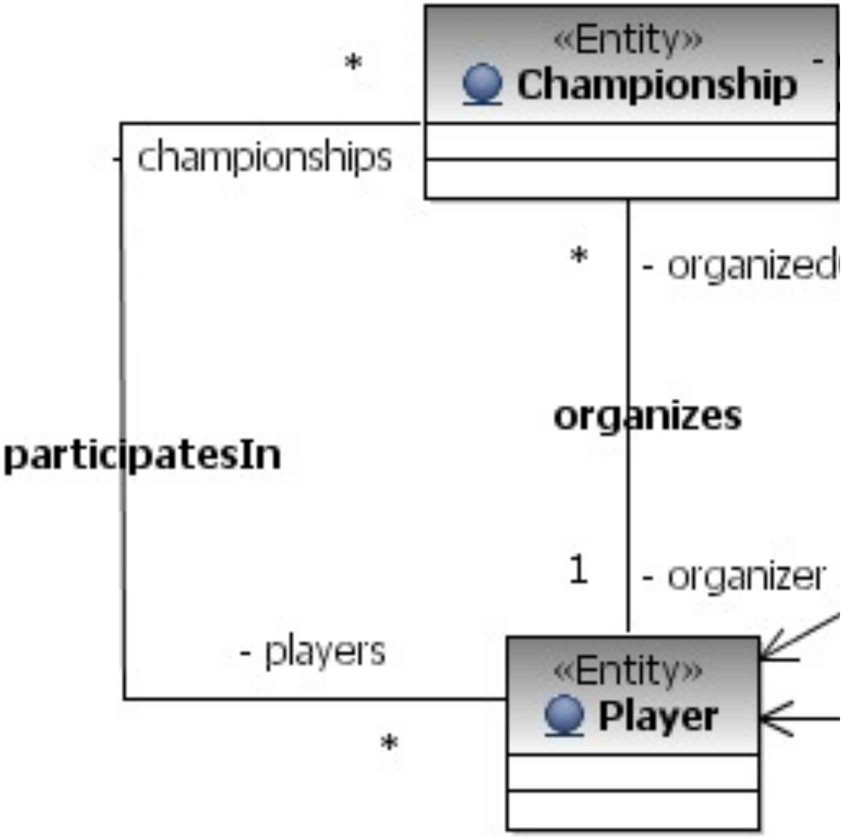
players:
    Championship -> Set(Player)

You do not need to write such constraints in OCL!

Navigation along roles

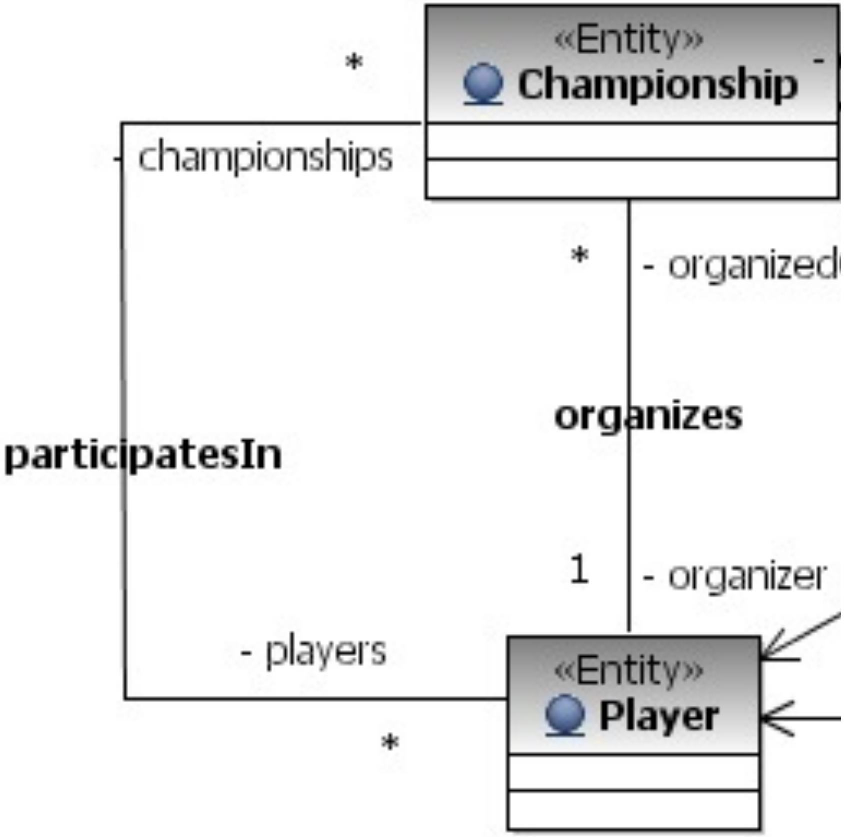


Navigation along roles



- Multiplicity 0..1

Navigation along roles

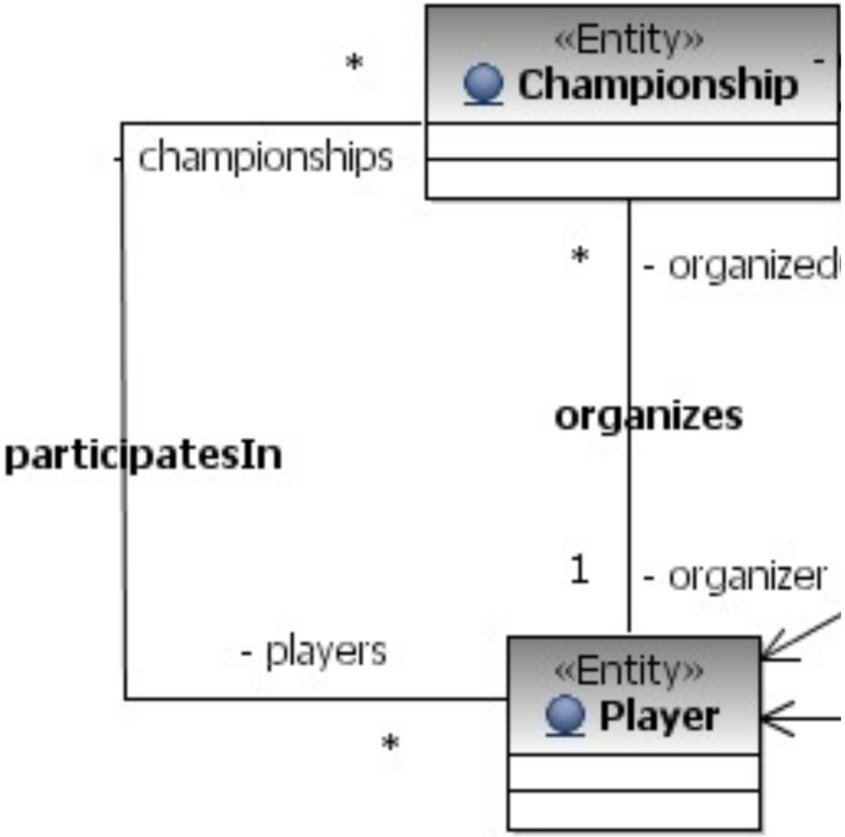


- Multiplicity 0..1

  context Championship inv:

  self.organizer.birth > 1976

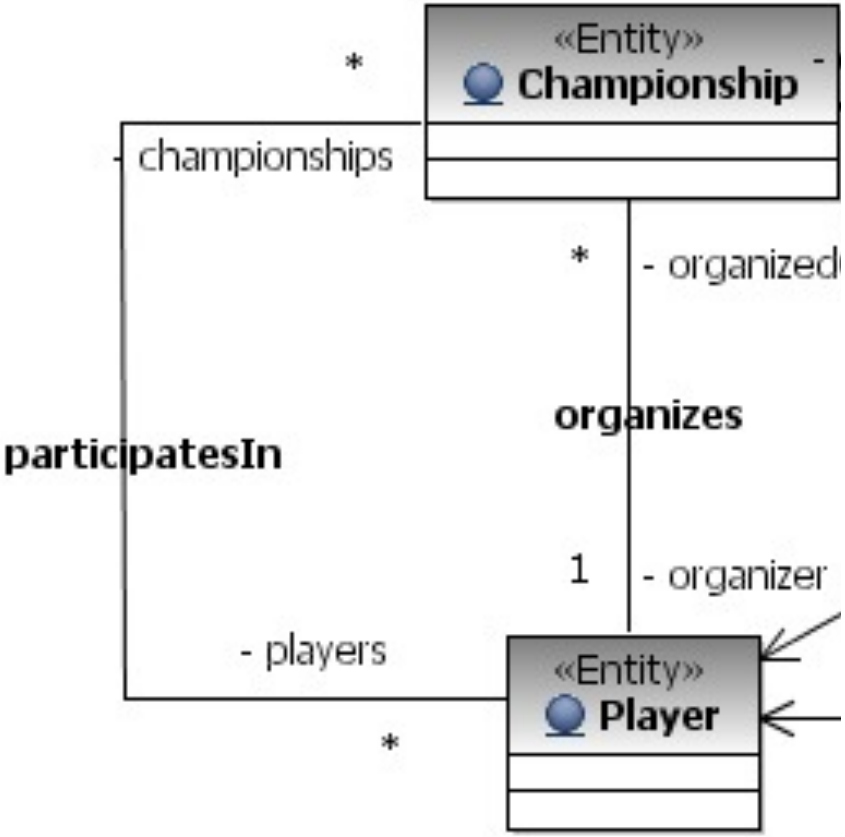
Navigation along roles



- Multiplicity 0..1

```
context Championship inv:
  self.organizer.birth > 1976
```

Navigation along roles



- Multiplicity 0..1

~~context Championship inv:~~
~~self.organizer.birth > 1976~~

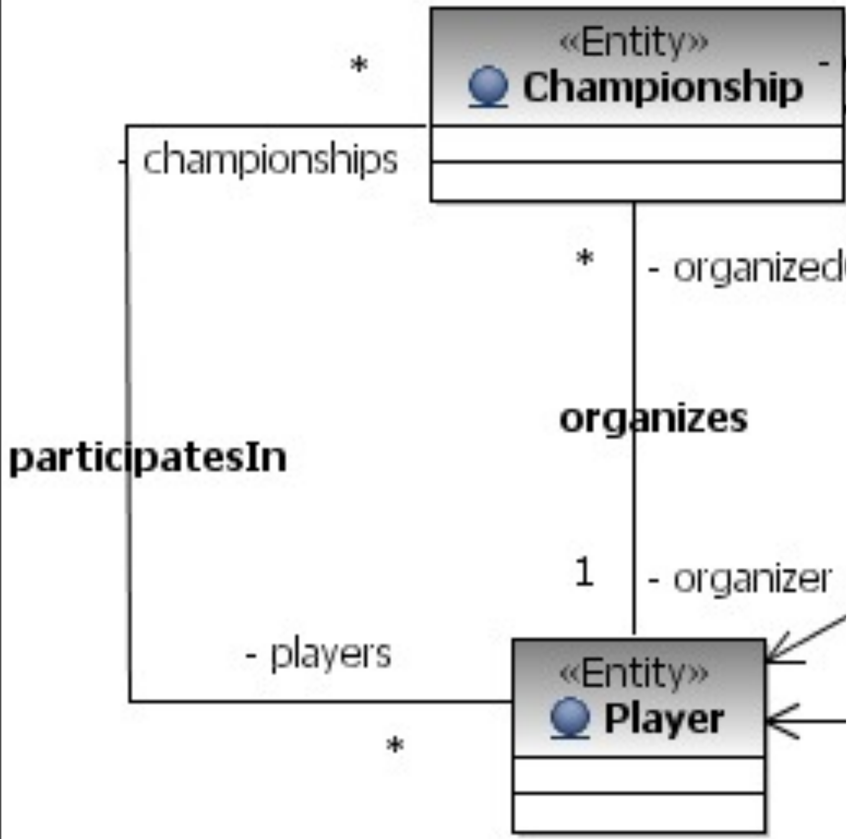
self.players results in a **collection**
self.players.birth: the coll. of birth years

Navigation along roles

Only attributes of an **object** can be compared with a value

- Multiplicity 0..1

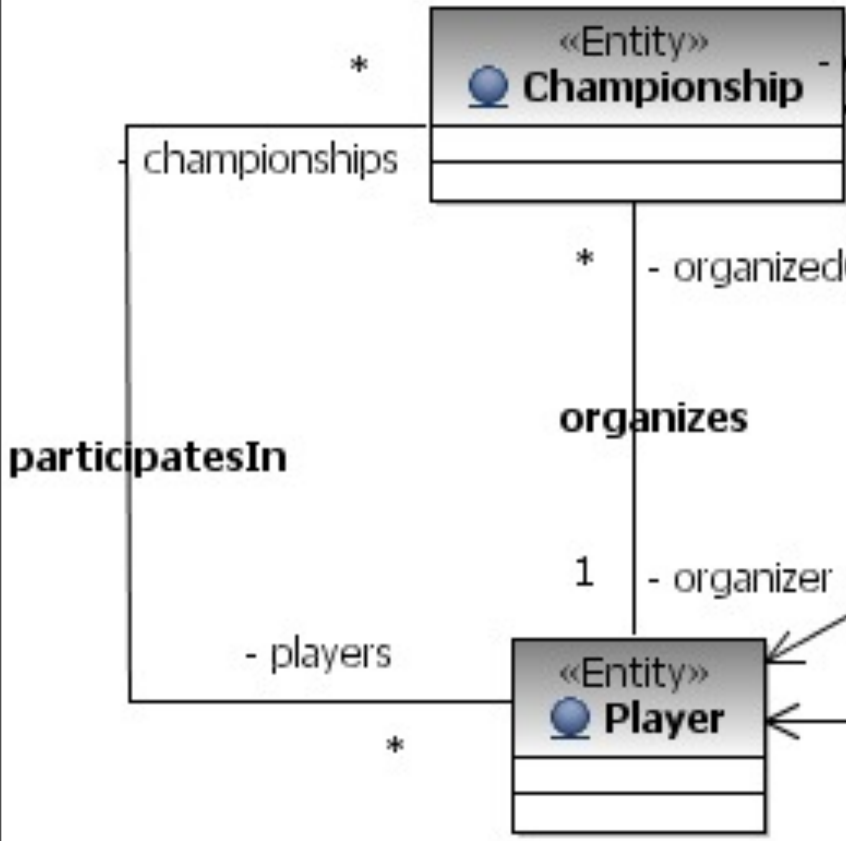
```
context Championship inv:
   self.organizer.birth > 1976
```



self.players results in a **collection**

self.players.birth: the coll. of birth years

Navigation along roles



Only attributes of an **object** can be compared with a value

- Multiplicity 0..1

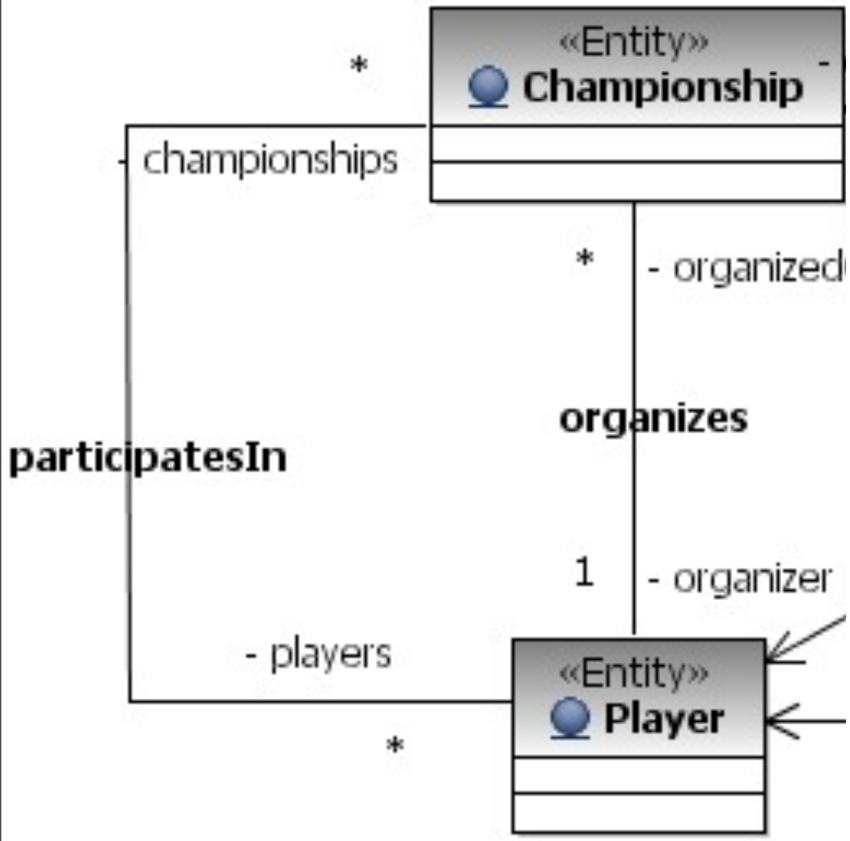
```
context Championship inv:
   self.organizer.birth > 1976
```

- Multiplicity * (many)

`self.players` results in a **collection**

`self.players.birth`: the coll. of birth years

Navigation along roles



Only attributes of an **object** can be compared with a value

- Multiplicity 0..1

  context Championship inv:
  self.organizer.birth > 1976

- Multiplicity * (many)

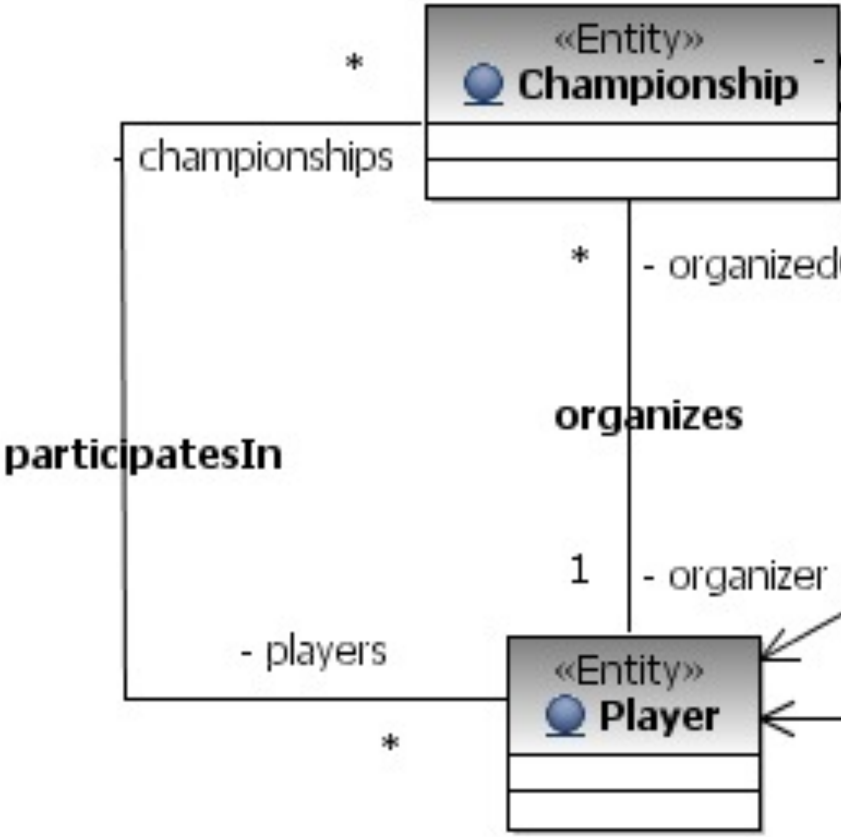
  ~~context Championship inv:~~
  ~~self.players.birth > 1976~~

self.players results in a **collection**

self.players.birth: the coll. of birth years

Navigation along roles

Only attributes of an **object** can be compared with a value



- Multiplicity 0..1

```
context Championship inv:
    self.organizer.birth > 1976
```

- Multiplicity * (many)

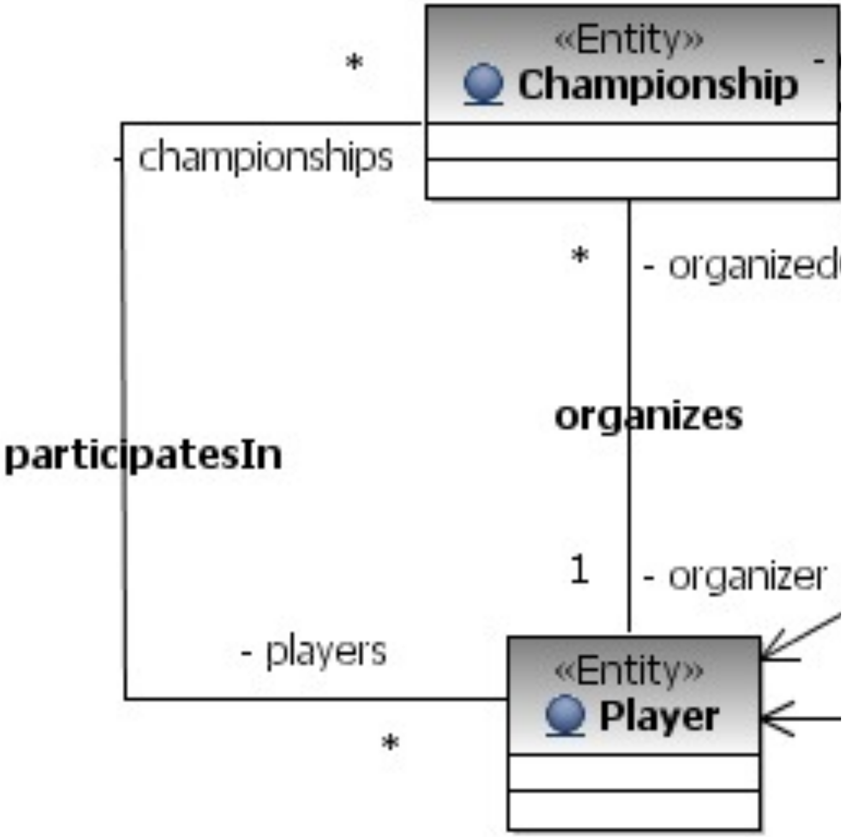
~~context Championship inv:~~
~~self.players.birth > 1976~~

self.players results in a **collection**

self.players.birth: the coll. of birth years

Navigation along roles

Only attributes of an **object** can be compared with a value



- Multiplicity 0..1

  context Championship inv:
  self.organizer.birth > 1976

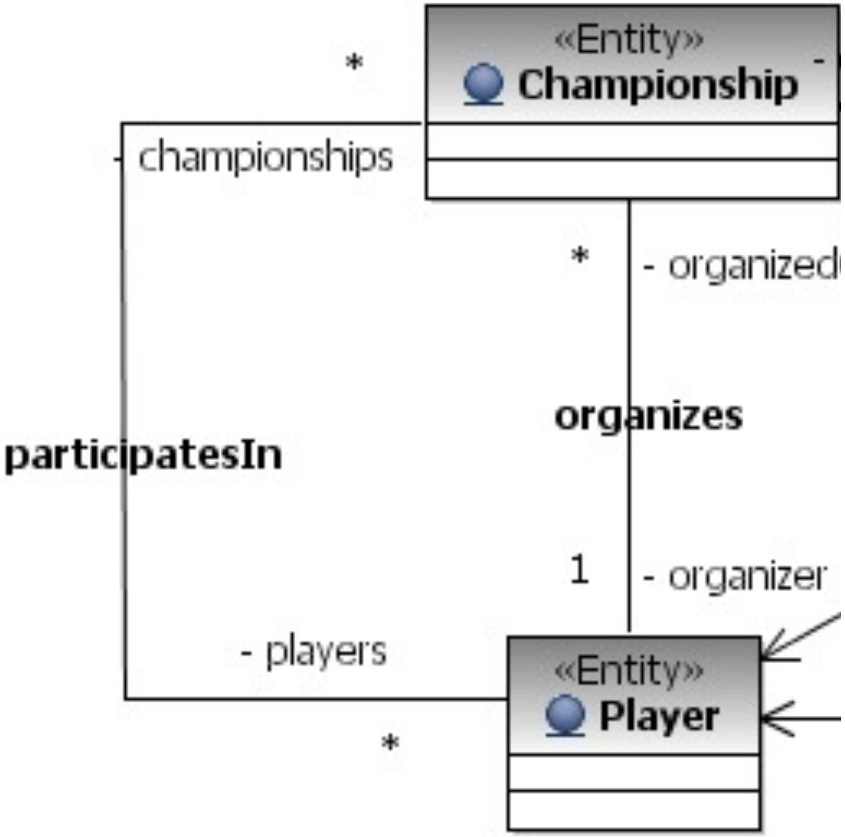
- Multiplicity * (many)

  ~~context Championship inv:~~
  ~~self.players.birth > 1976~~

self.players results in a **collection**

self.players.birth: the coll. of birth years

Navigation along roles



- Multiplicity 0..1

```
context Championship inv:
    self.organizer.birth > 1976
```

Only attributes of an **object** can be compared with a value

- Multiplicity * (many)

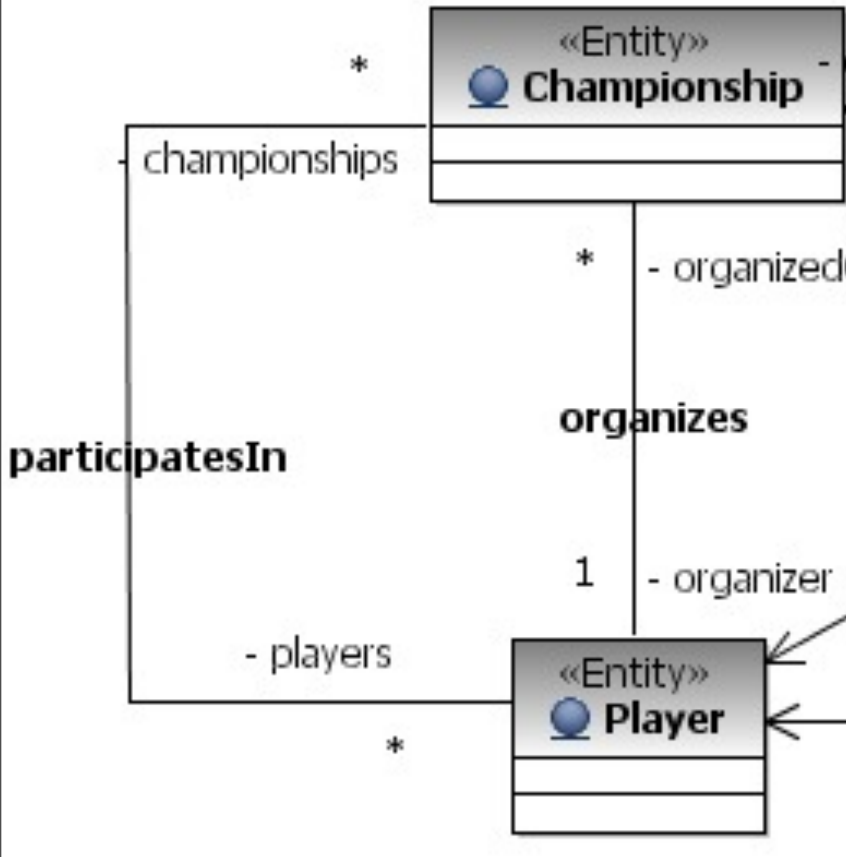
~~context Championship inv:~~
~~self.players.birth > 1976~~

self.players results in a **collection**

self.players.birth: the coll. of birth years

Navigation along roles

Only attributes of an **object** can be compared with a value



- Multiplicity 0..1

  context Championship inv:
  self.organizer.birth > 1976

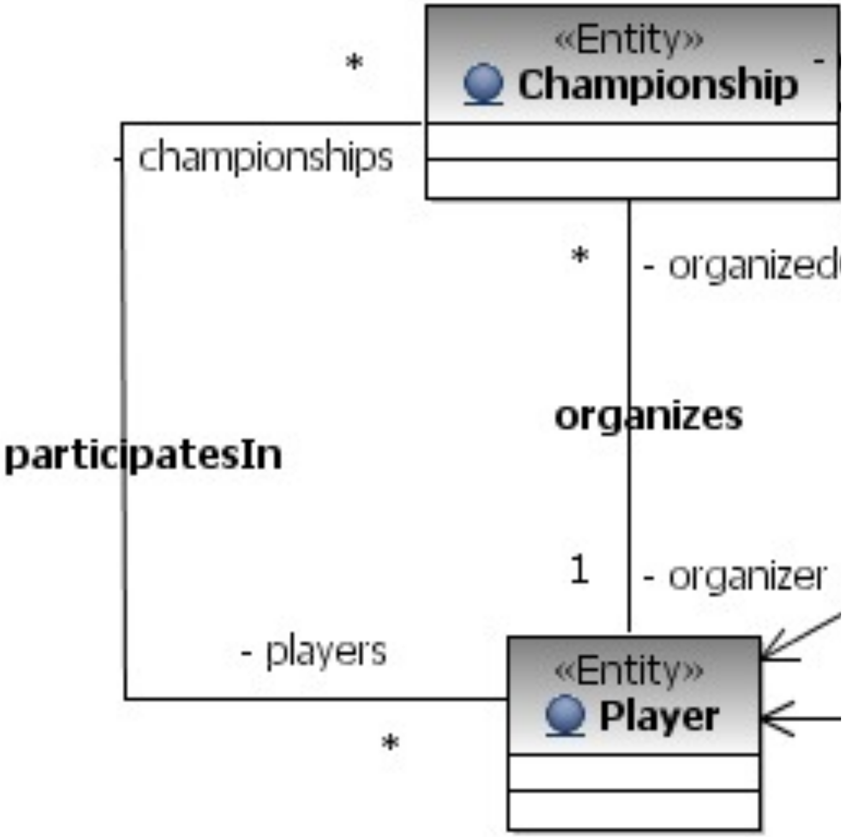
- Multiplicity * (many)

  ~~context Championship inv:~~
  ~~self.players.birth > 1976~~

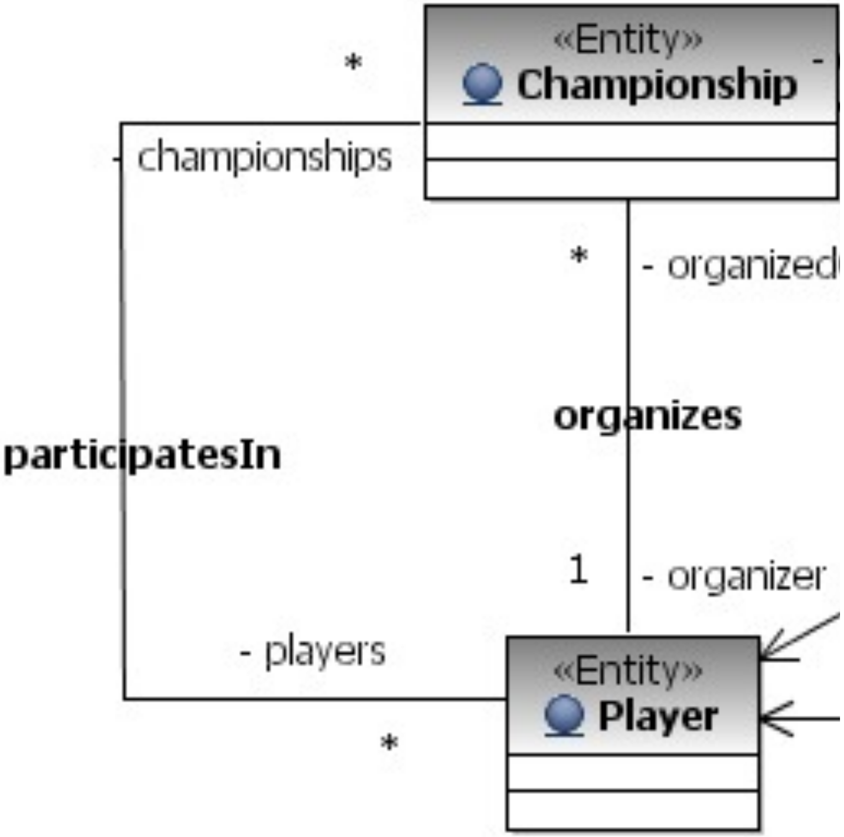
self.players results in a **collection**
self.players.birth: the coll. of birth years

context Championship inv:
self.players-> …
(operations on collections)

Consistency of bidirectional associations

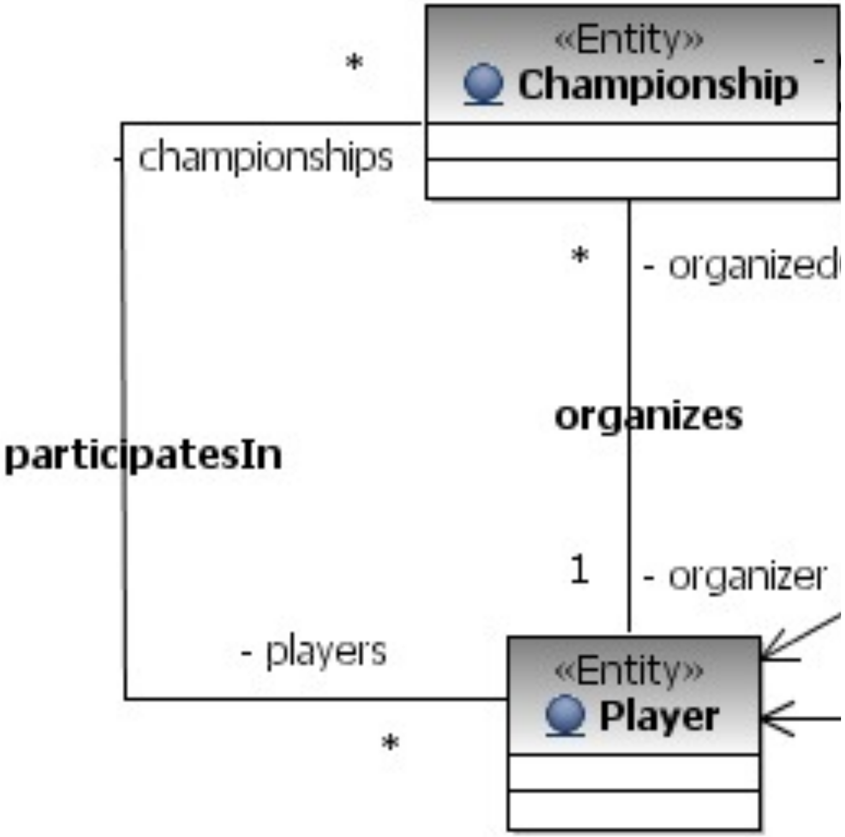


Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

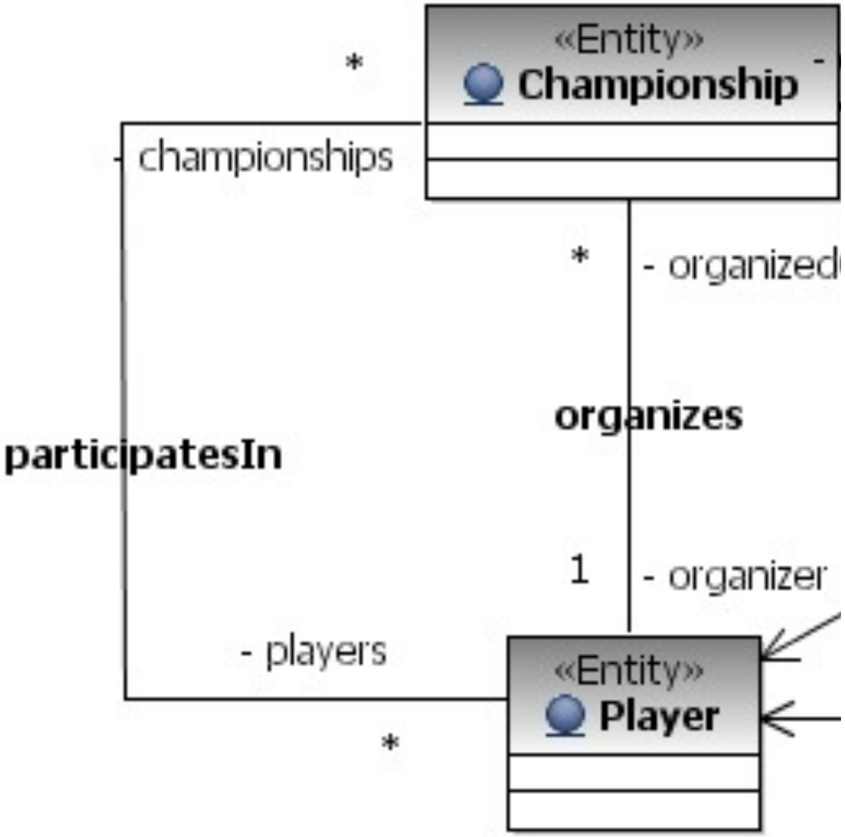
Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

Collection = Single object
Such an equality is invalid

Consistency of bidirectional associations



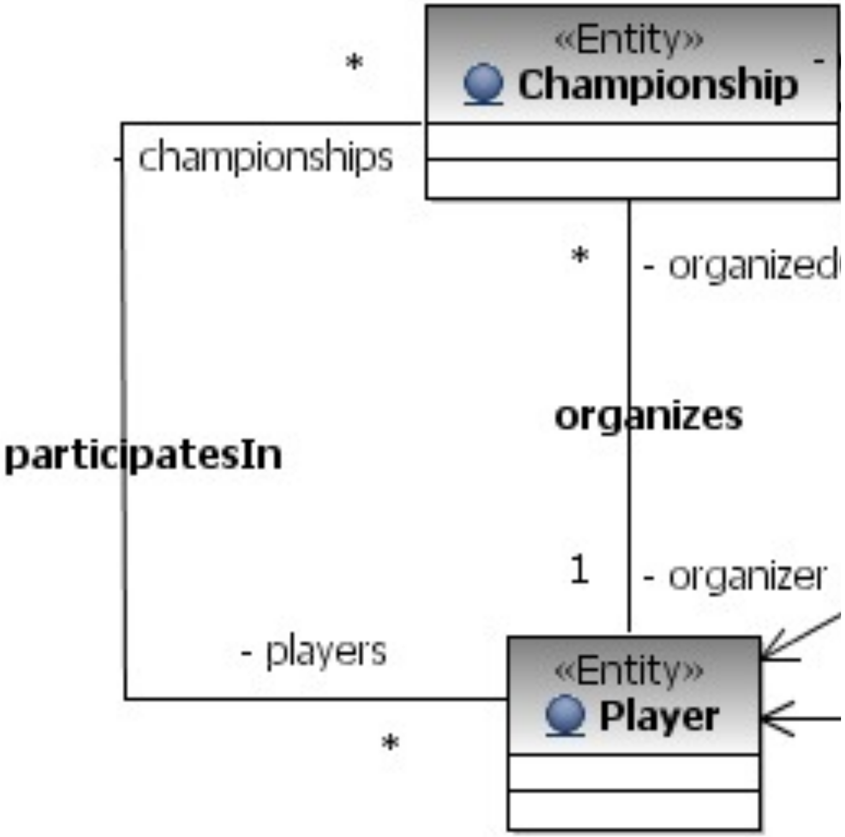
- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Championship inv:~~

~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

Consistency of bidirectional associations



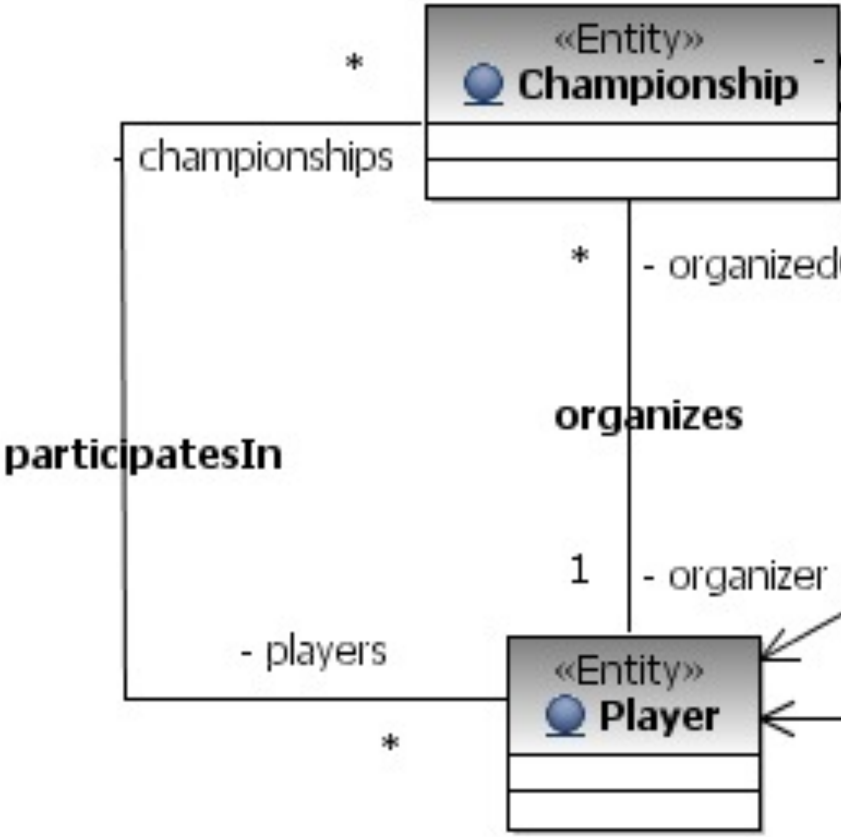
- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Championship inv:~~

~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

Consistency of bidirectional associations



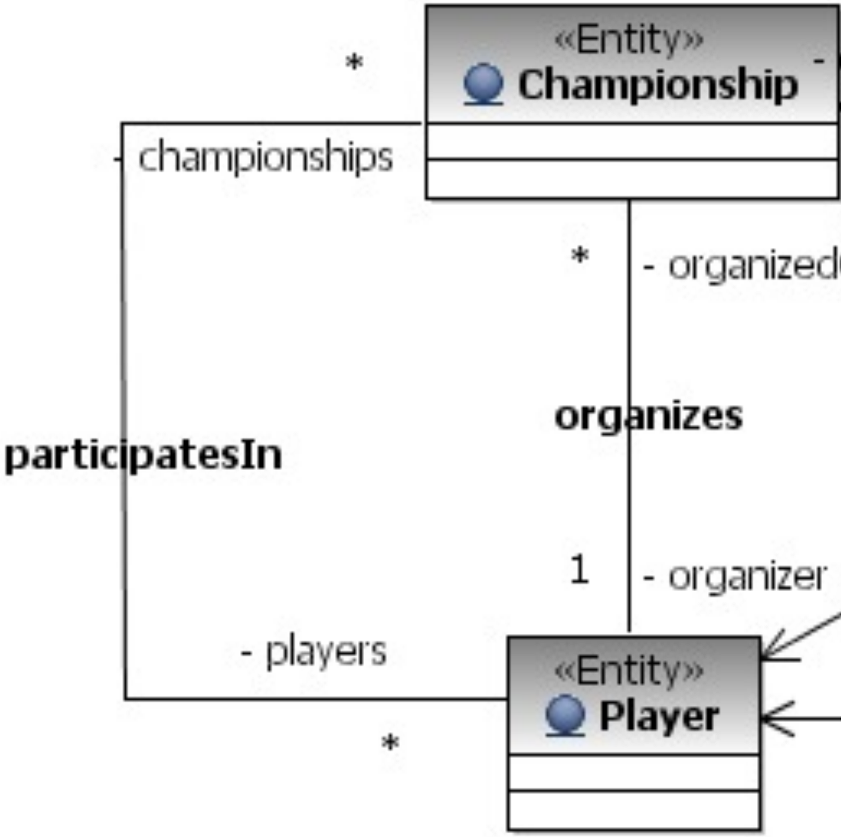
- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Championship inv:~~

~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

Consistency of bidirectional associations



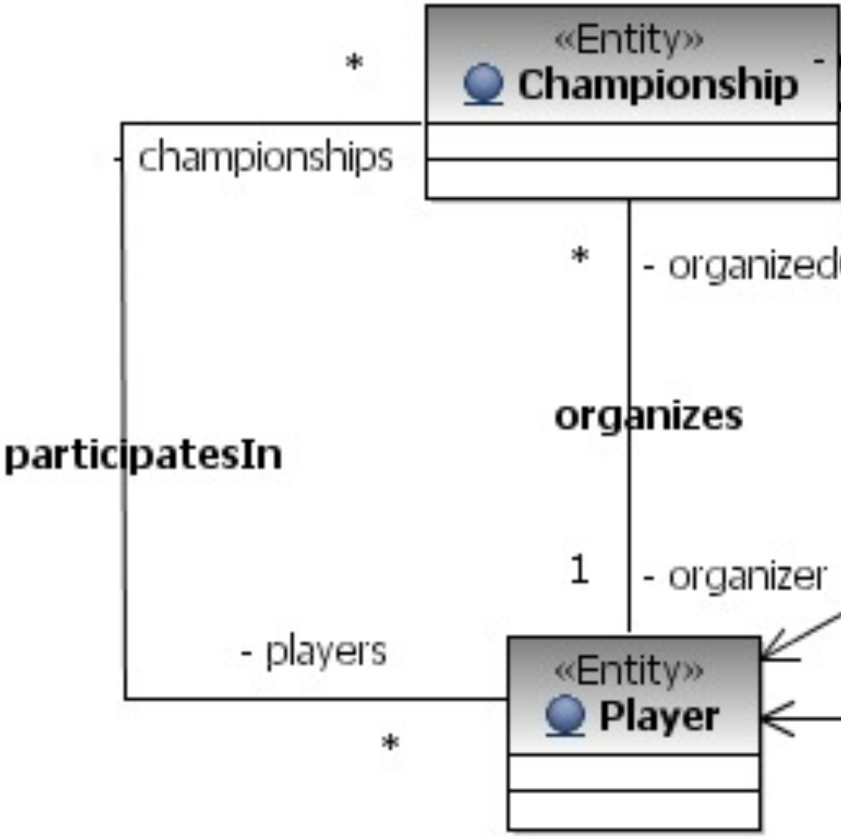
- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Championship inv:~~
~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

```
context Championship inv:
   self.organizer.organized->
   includes(self)
```

Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

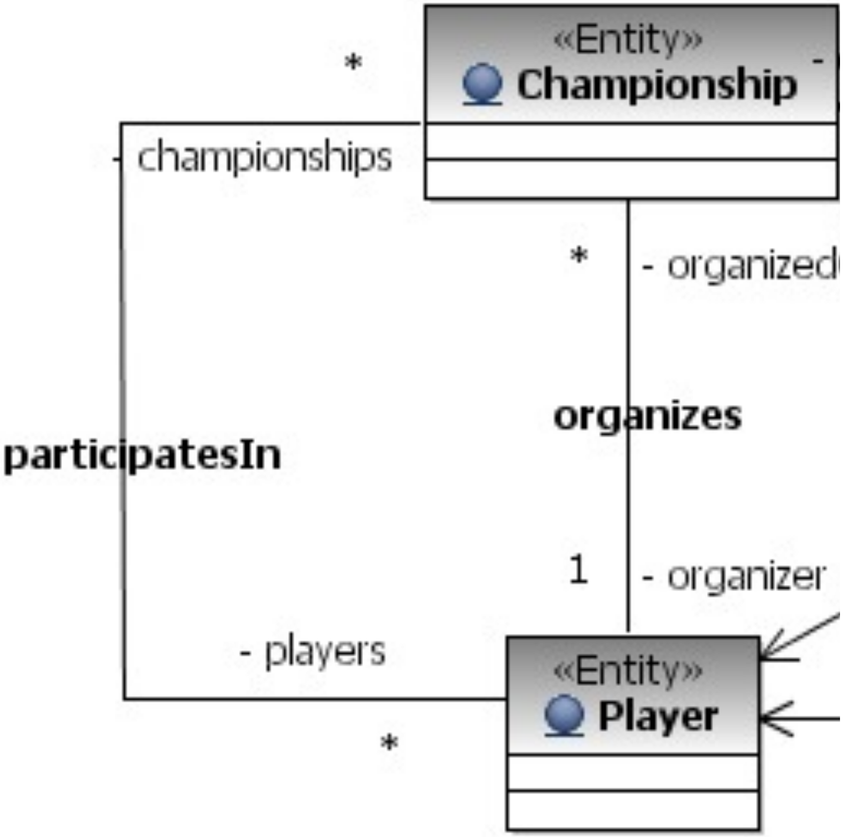
~~context Championship inv:~~
~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

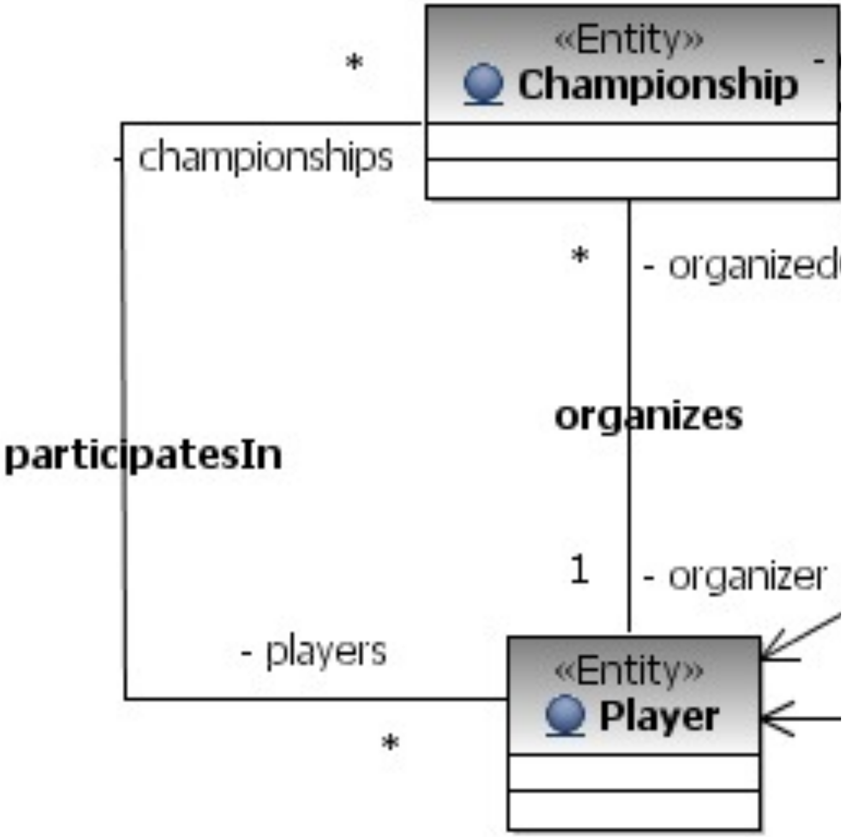
```
context Championship inv:
   self.organizer.organized->
   includes(self)
```

Coll->includes(e):
Tests collection membership: **e ∈Coll**

Consistency of bidirectional associations

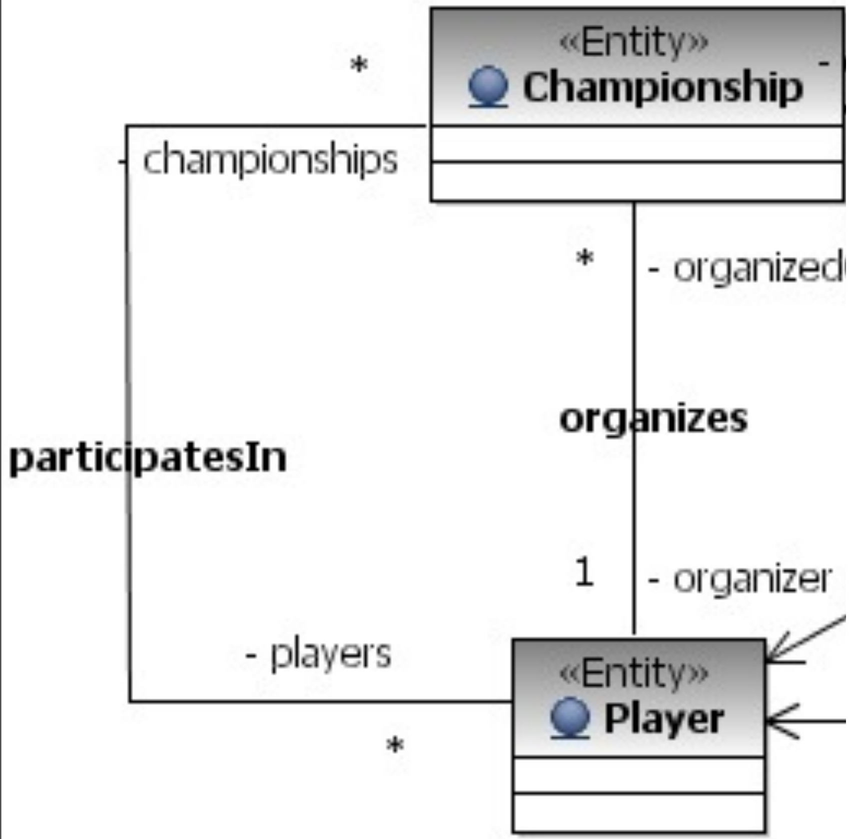


Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

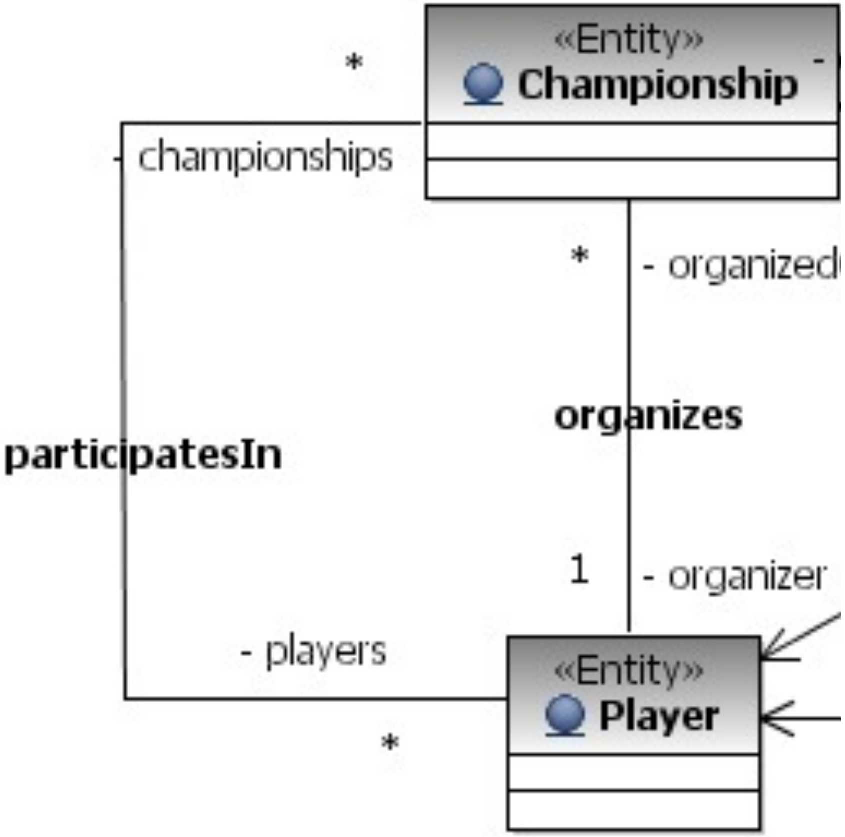
Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

Incorrect: Constraint is prescribed **for all** champs

Consistency of bidirectional associations

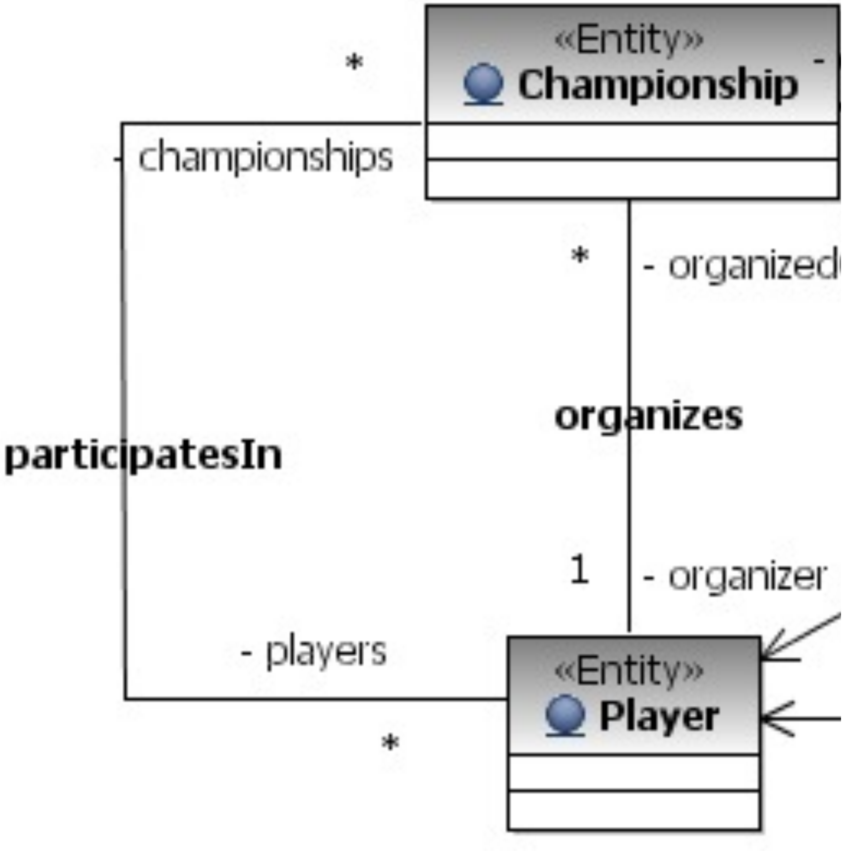


- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Player inv:~~
~~self.organized->exists(c |~~
~~c.organizer = self)~~

Incorrect: Constraint is prescribed **for all** champs

Consistency of bidirectional associations

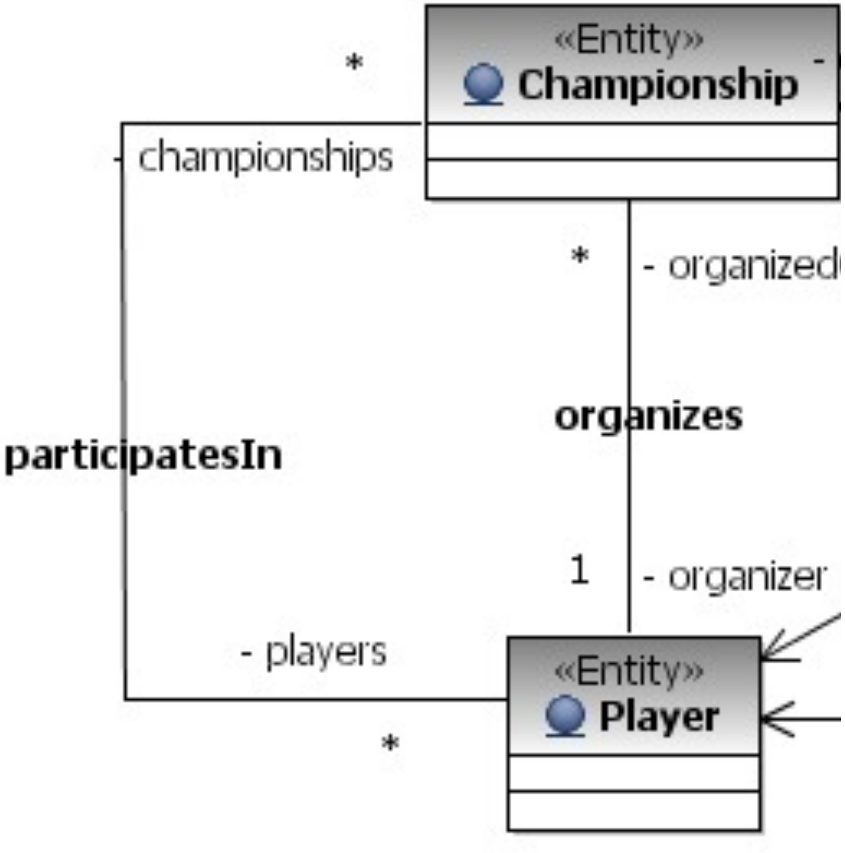


- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Player inv:~~
~~self.organized->exists(c |~~
~~c.organizer = self)~~

Incorrect: Constraint is prescribed **for all** champs

Consistency of bidirectional associations

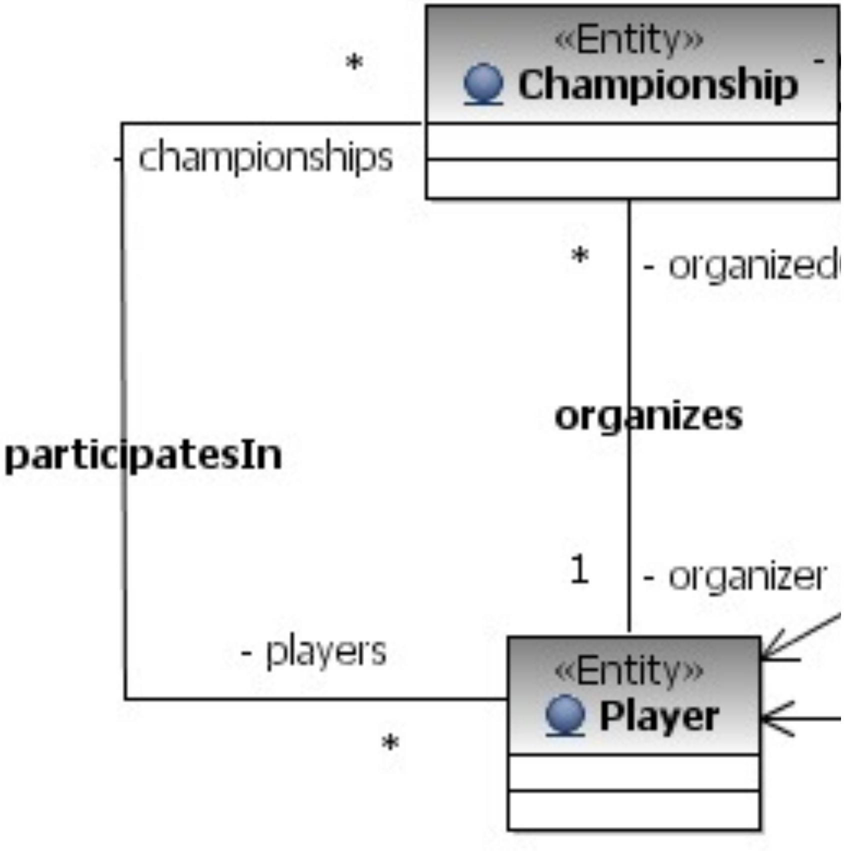


- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Player inv:~~
~~self.organized->exists(c |~~
~~c.organizer = self)~~

Incorrect: Constraint is prescribed **for all** champs

Consistency of bidirectional associations



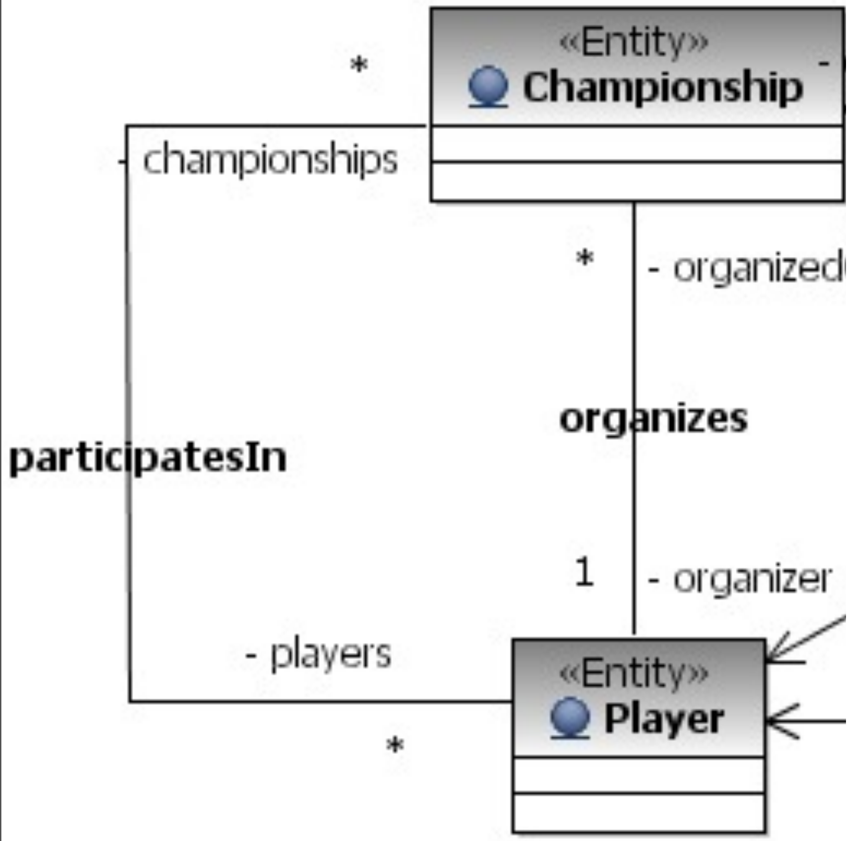
- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Player inv:~~
~~self.organized->exists(c |~~
~~c.organizer = self)~~

Incorrect: Constraint is prescribed **for all** champs

```
context Player inv:
   self.organized->forAll(c |
   c.organizer = self)
```

Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

~~context Player inv:~~

~~self.organized->exists(c |~~

c.organizer = self)

Incorrect: Constraint is prescribed **<u>for all</u>** champs

context Player inv:

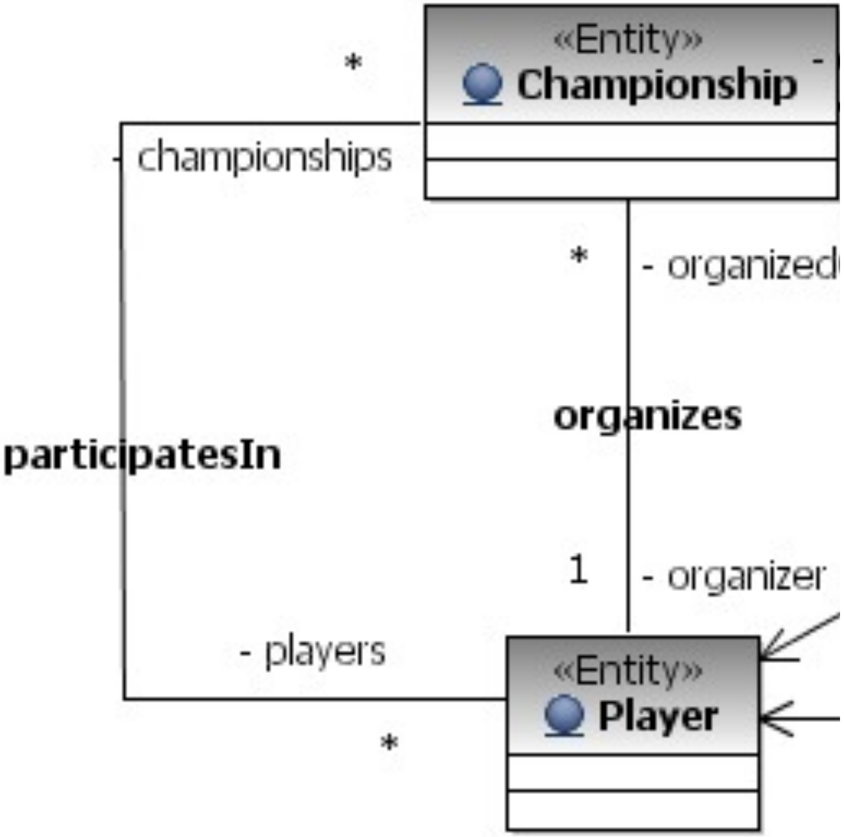
self.organized->fo ll(c |

c.org

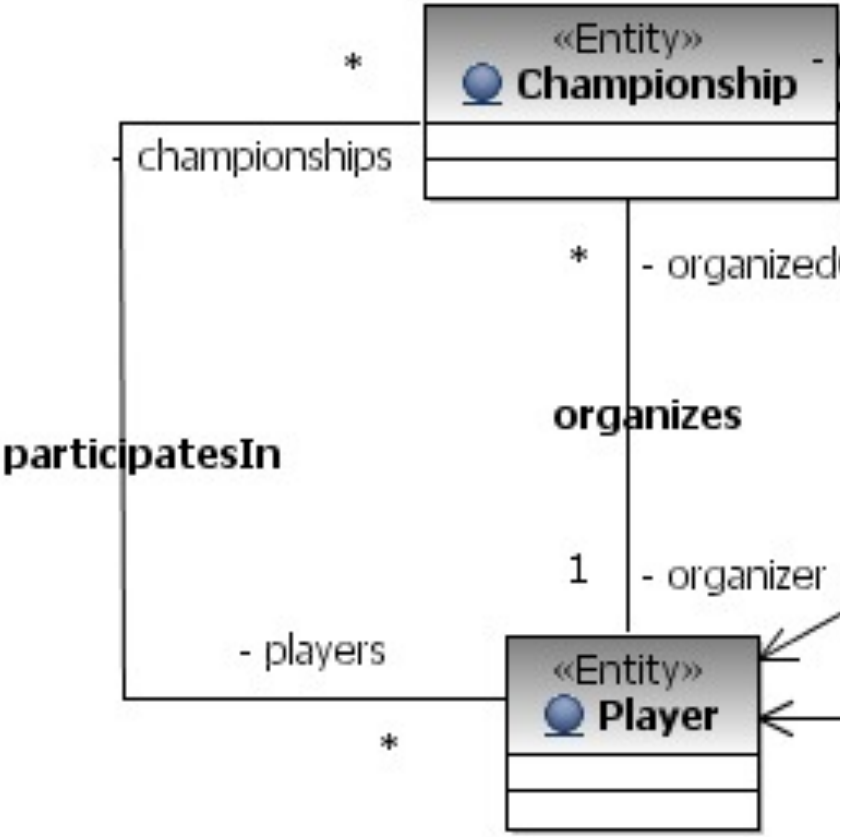
Coll->forAll(e|cond(e))

Quantifiers can only be applied to collections

Consistency of bidirectional associations

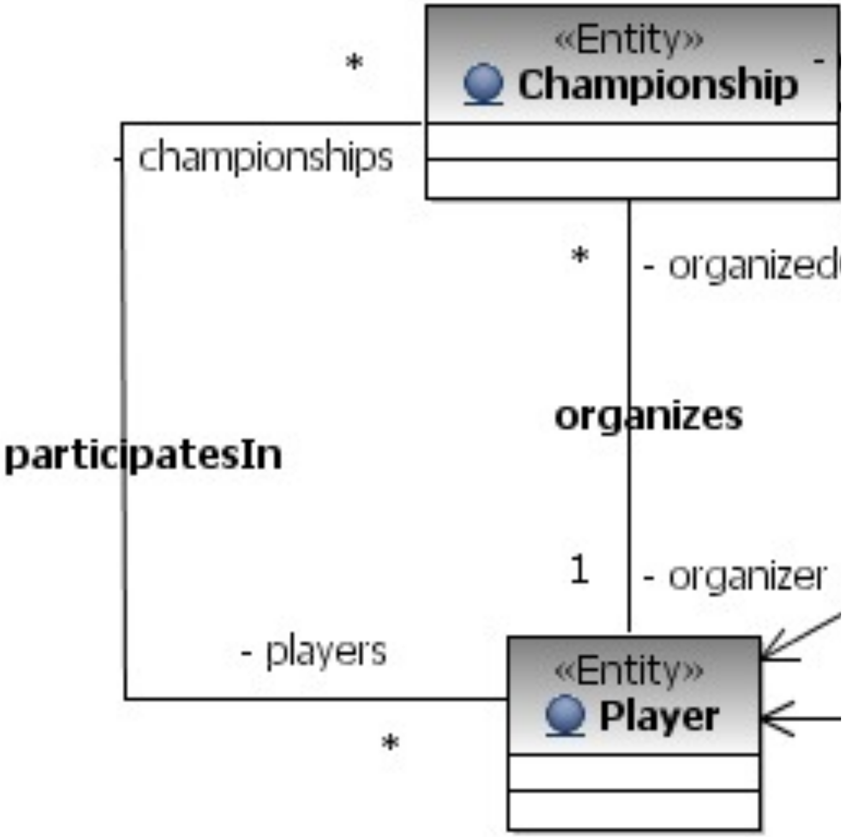


Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

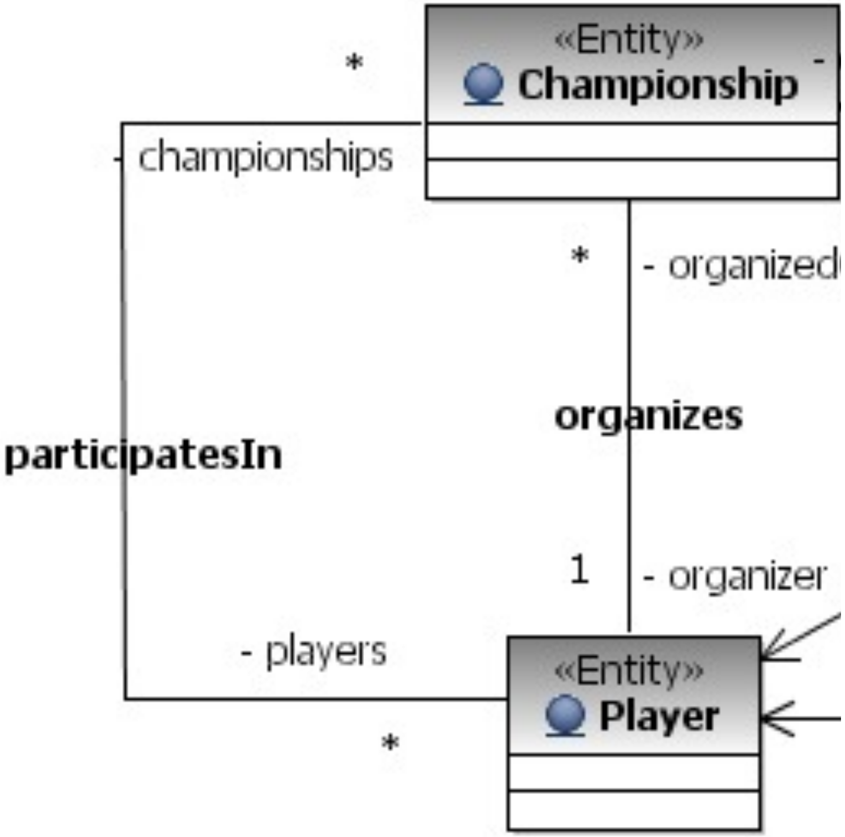
Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

```
context Championship inv:
   self.players->forall(p |
   p.championships-> includes
   (self))
```

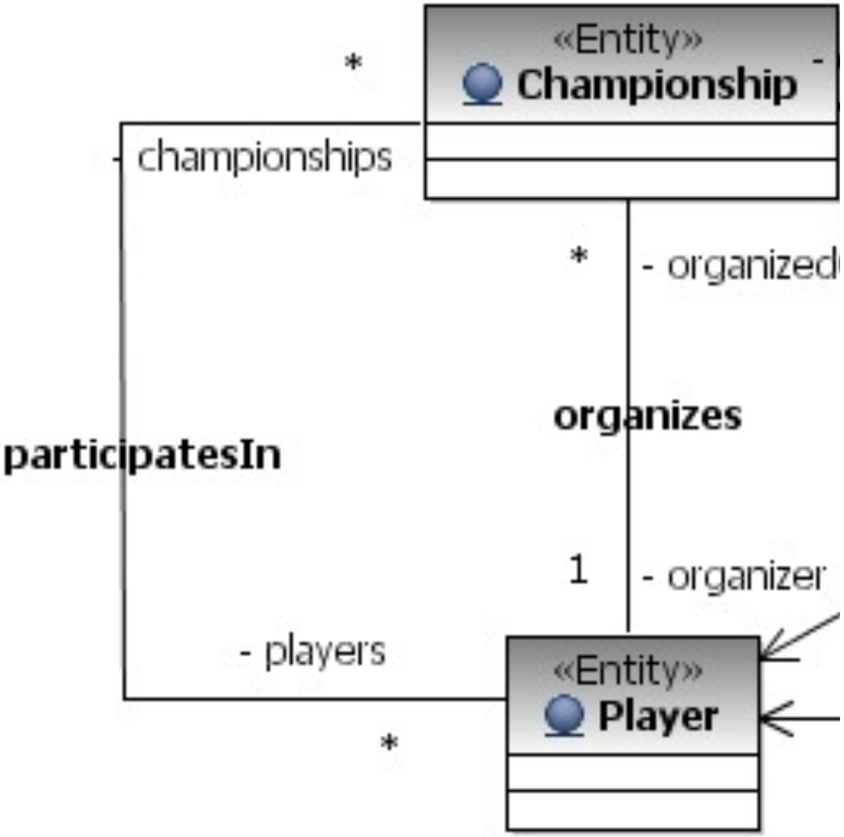
Consistency of bidirectional associations



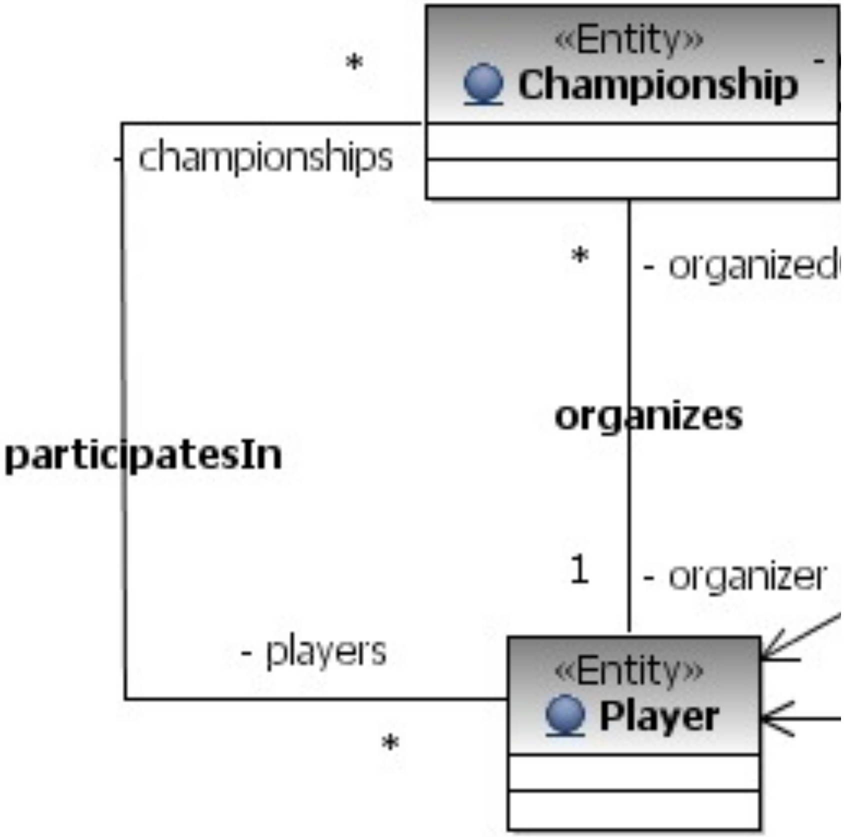
- If a bidirectional association exists between two objects then it is navigable from both directions

```
context Championship inv:
   self.players->forall(p |
   p.championships-> includes
   (self))
context Player inv:
   self.championships->forall(c
   | c.players -> includes(self))
```

Consistency of bidirectional associations

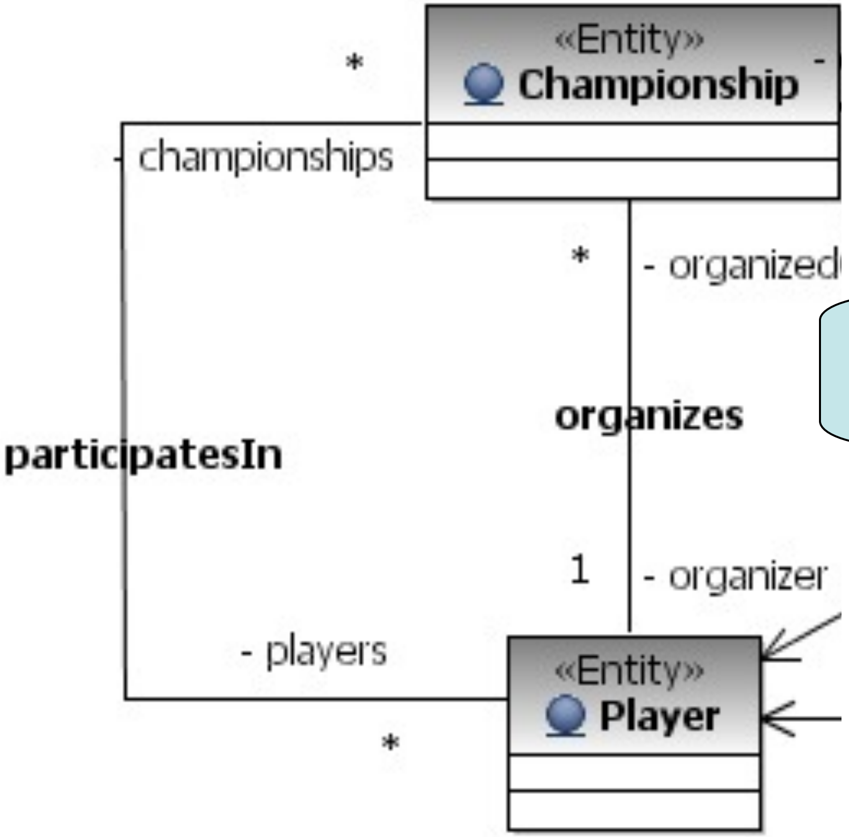


Consistency of bidirectional associations



- The organizer of the championship organizes at least one championship

Consistency of bidirectional associations

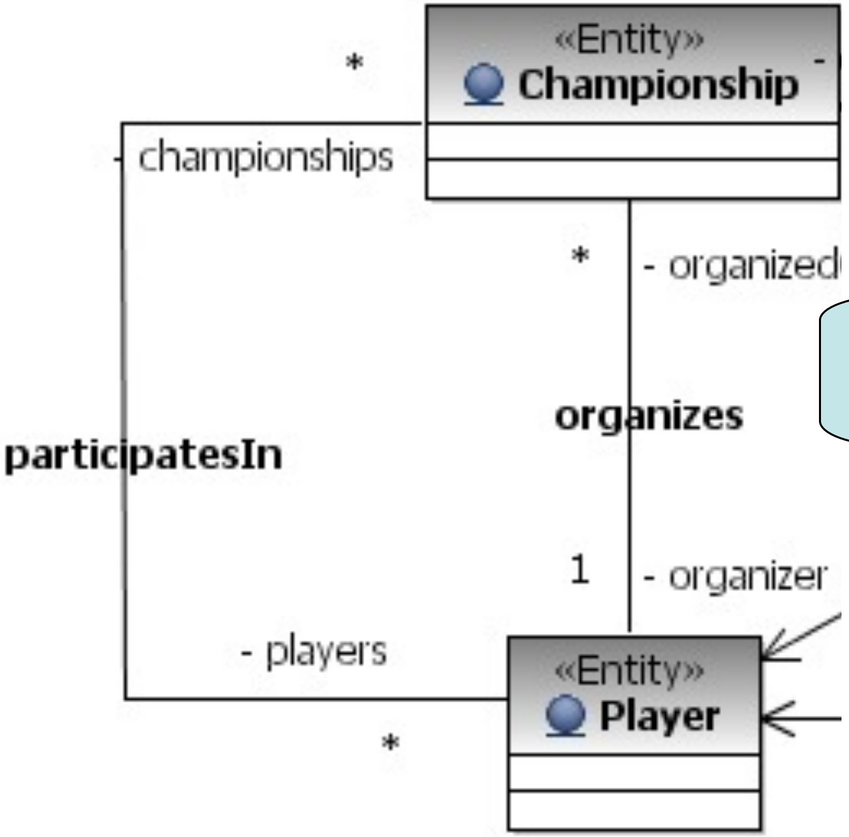


- The organizer of the championship organizes at least one championship

Context should be Championship

No player is forced to organize a champs

Consistency of bidirectional associations



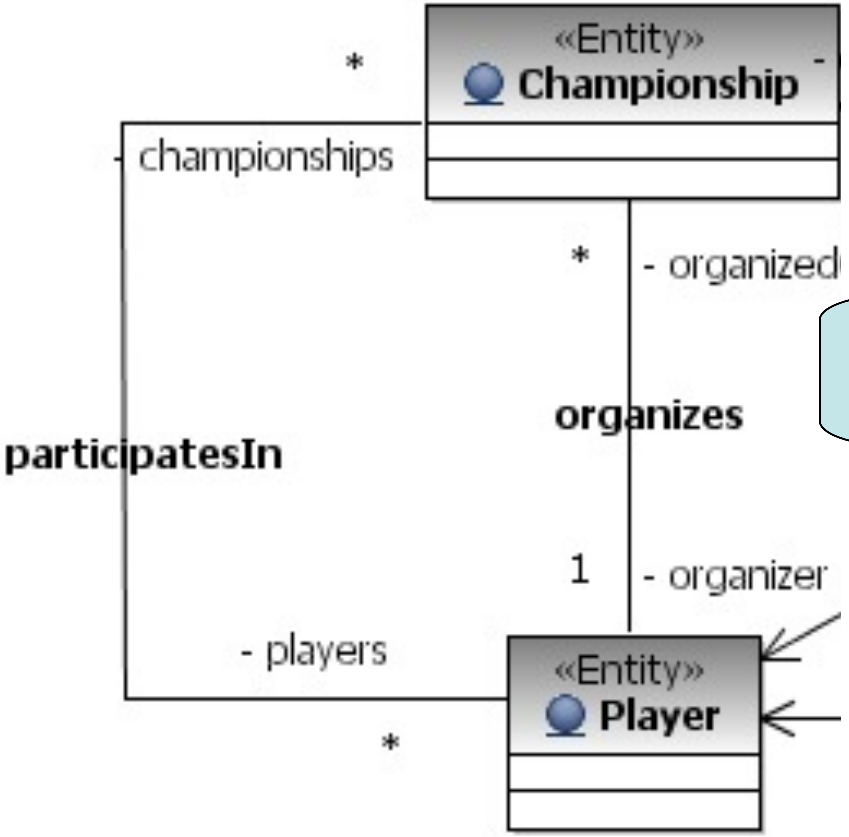
- The organizer of the championship organizes at least one championship

```
~~context Player inv:~~
~~self.organized->size > 0~~
```

Context should be Championship

No player is forced to organize a champs

Consistency of bidirectional associations



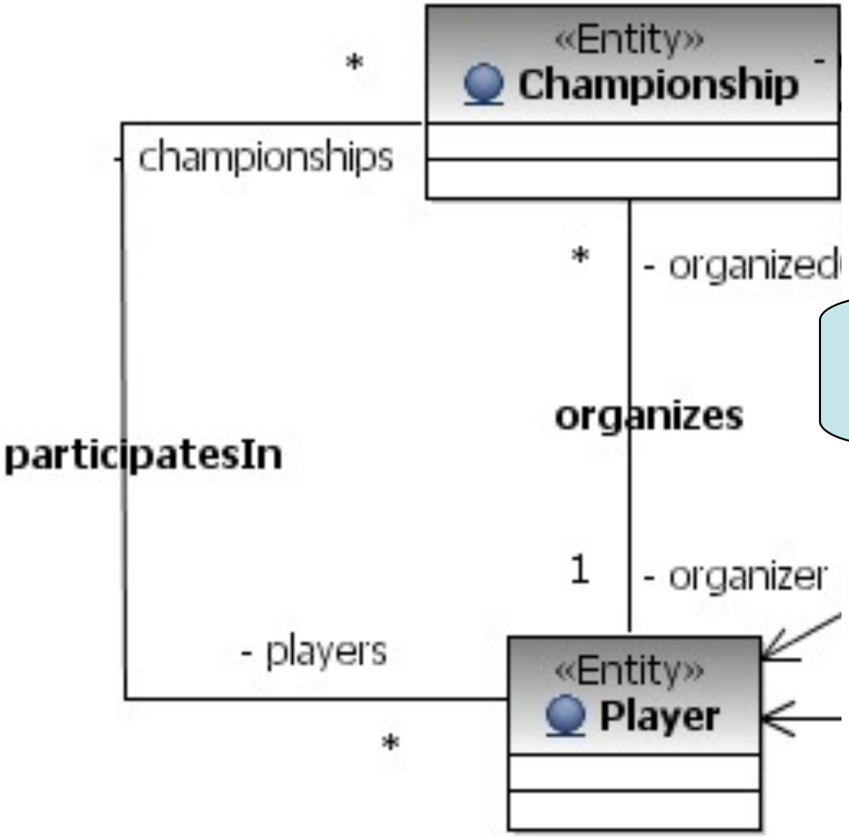
- The organizer of the championship organizes at least one championship

```
~~context Player inv:~~
~~self.organized->size > 0~~
```

Context should be Championship

No player is forced to organize a champs

Consistency of bidirectional associations



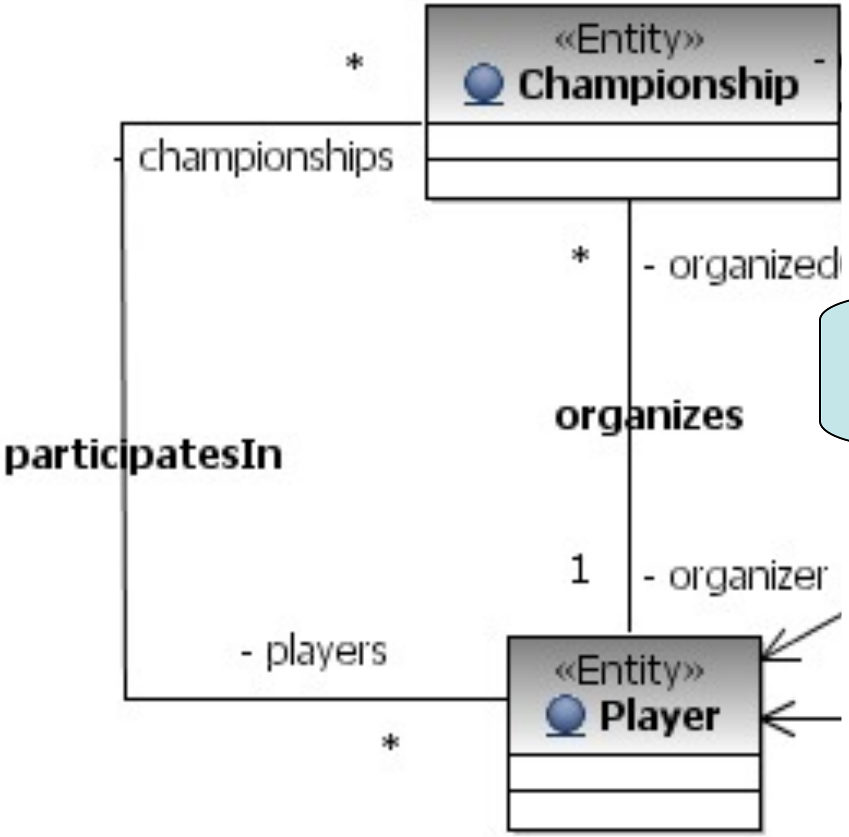
- The organizer of the championship organizes at least one championship

```
~~context Player inv:~~
~~self.organized->size > 0~~
```

Context should be Championship

No player is forced to organize a champs

Consistency of bidirectional associations



- The organizer of the championship organizes at least one championship

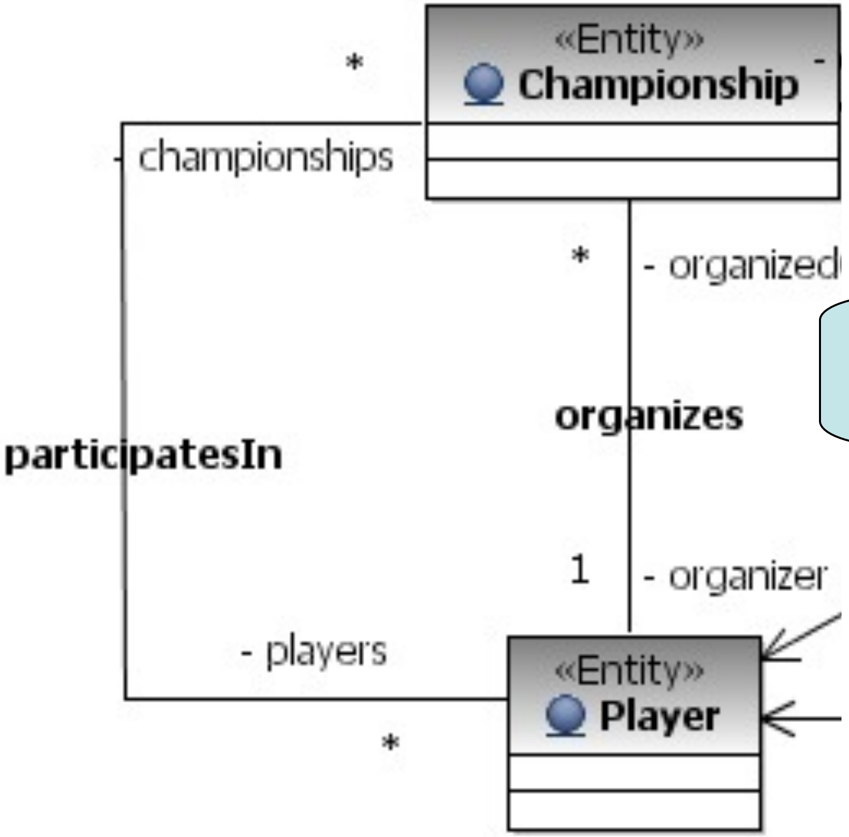
~~context Player inv:~~
~~self.organized->size > 0~~

Context should be Championship

No player is forced to organize a champs

```
context Championship inv:
   self.organizer.organized->
   size > 0
```

Consistency of bidirectional associations



- The organizer of the championship organizes at least one championship

~~context Player inv:~~
~~self.organized->size > 0~~

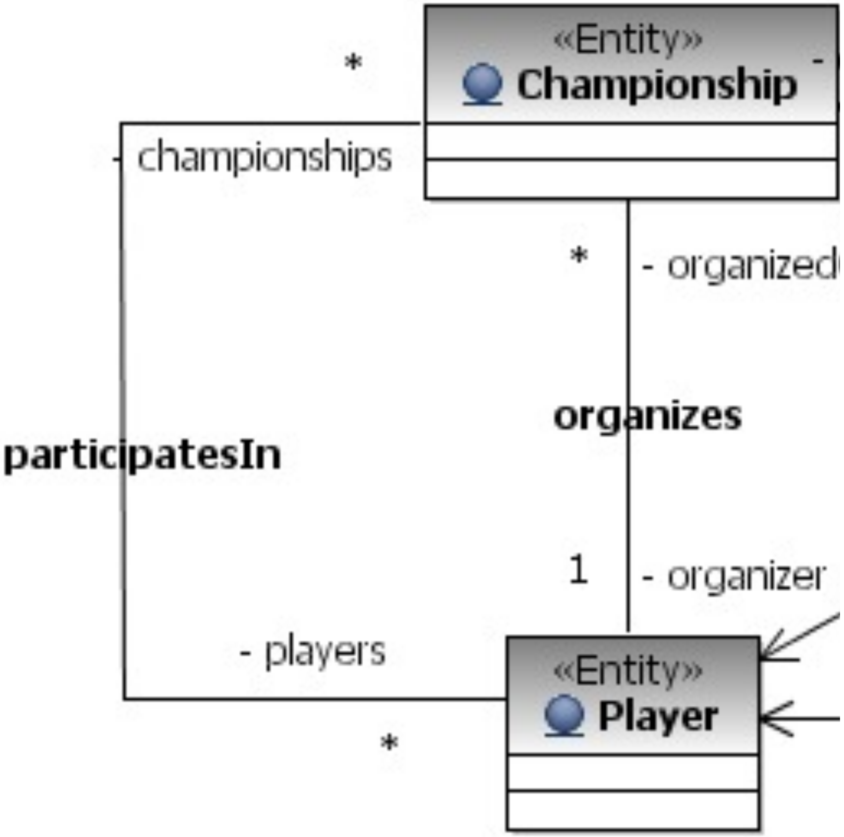
Context should be Championship

No player is forced to organize a champs

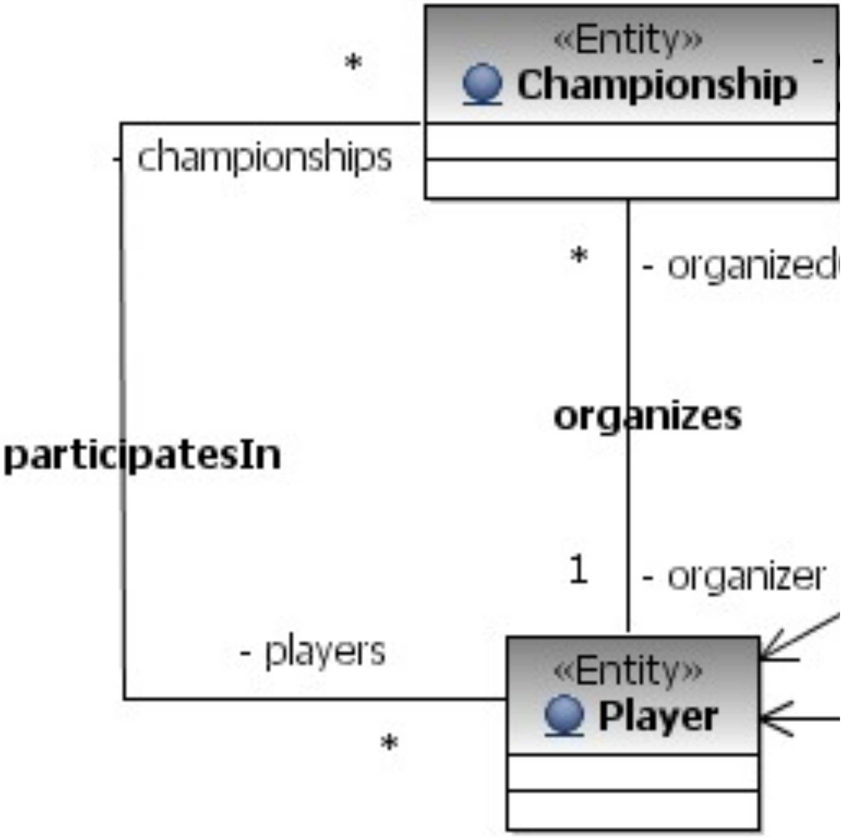
```
context Championship inv:
   self.organizer.organized->
   size > 0
```

```
context Championship inv:
   self.organizer.organized->
   notEmpty
```

Application specific constraints

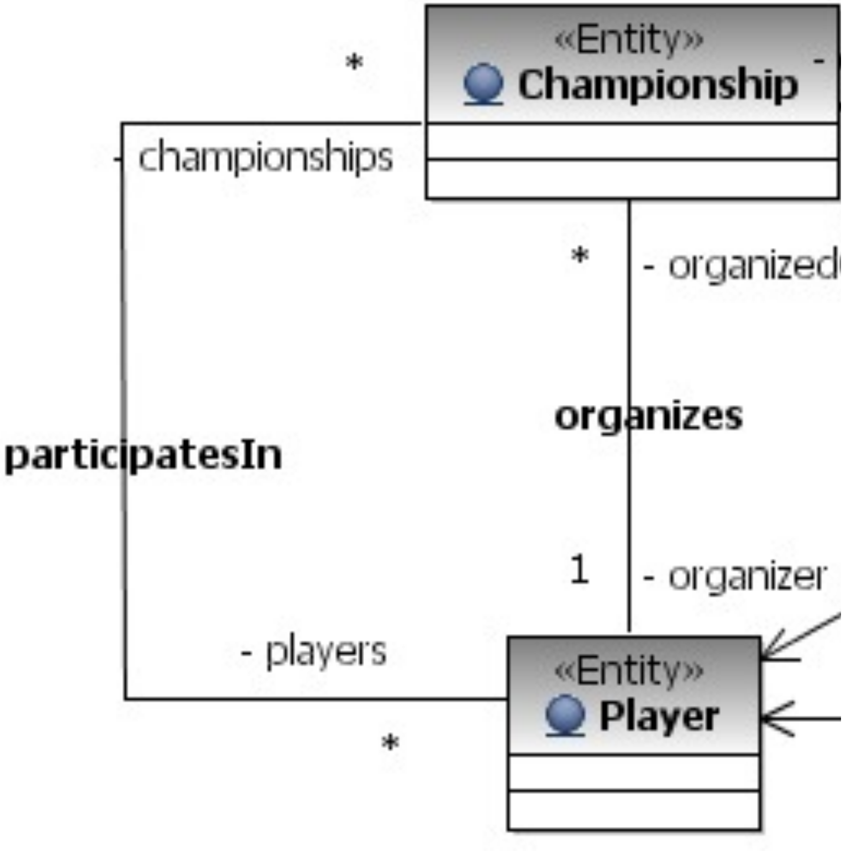


Application specific constraints



- A player is allowed to organize a single active championship at a time

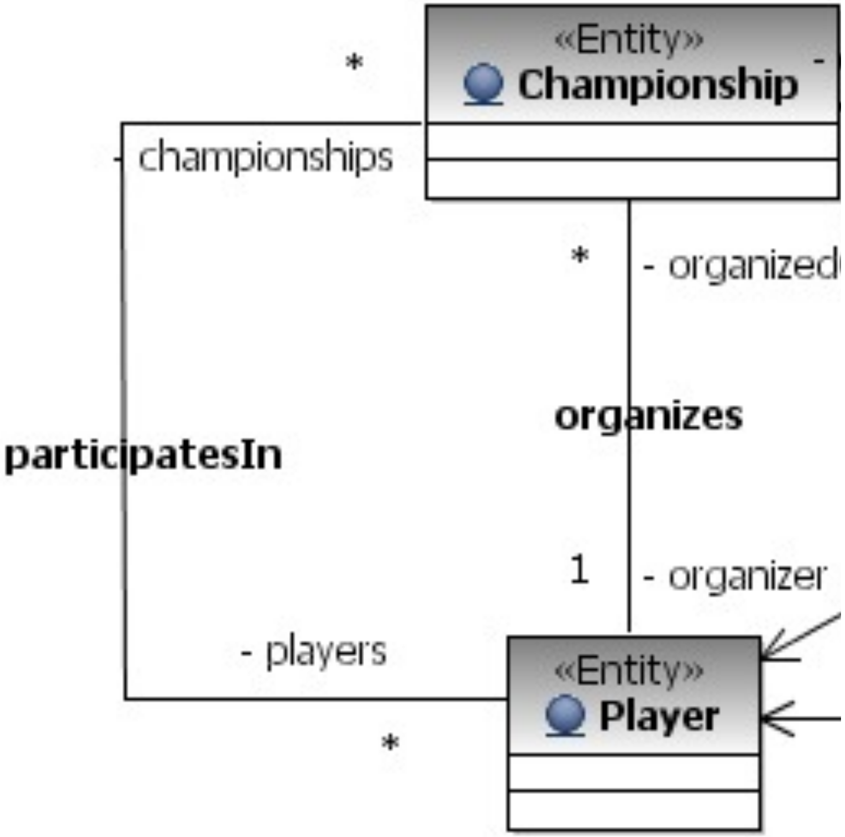
Application specific constraints



- A player is allowed to organize a single active championship at a time

```
context Player inv:
   self.organized->forall(c1, c2 |
   c1<>c2 implies
   (c1.status = ChS::closed or
    c1.status = ChS::cancelled) or
   (c2.status = ChS::closed or
    c2.status = ChS::cancelled))
```

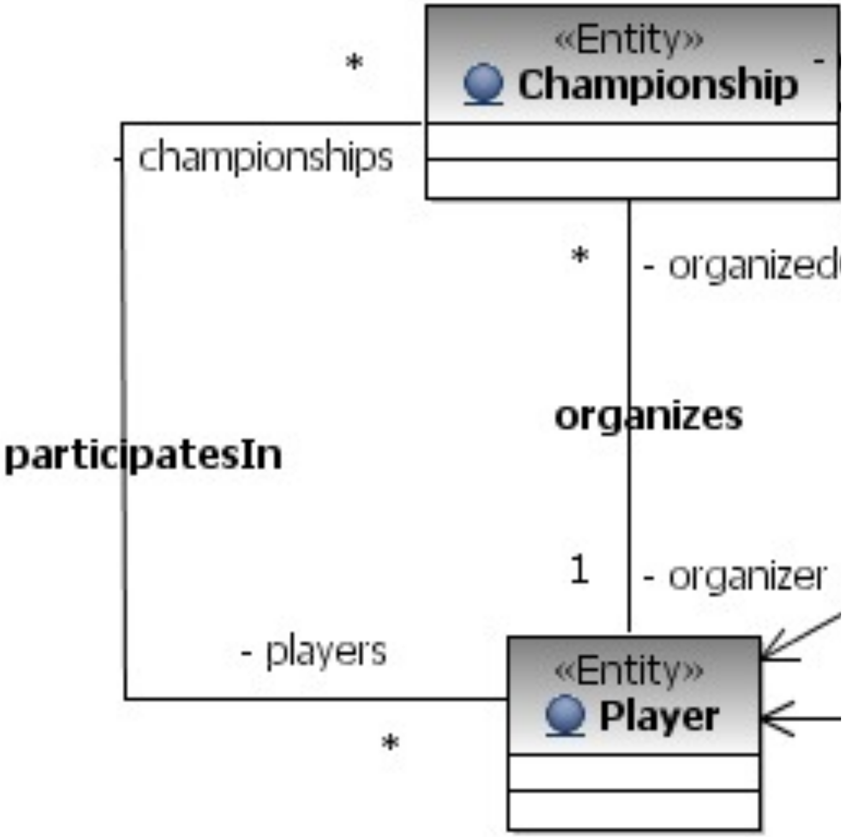
Application specific constraints



- A player is allowed to organize a single active championship at a time

```
context Player inv:
   self.organized->forall(c1, c2 |
   c1<>c2 implies
   (c1.status = ChS::closed or
    c1.status = ChS::cancelled) or
   (c2.status = ChS::closed or
    c2.status = ChS::cancelled))
context Player inv:
   self.organized->select(c |
   c.status = ChS::announced or
```

Application specific constraints

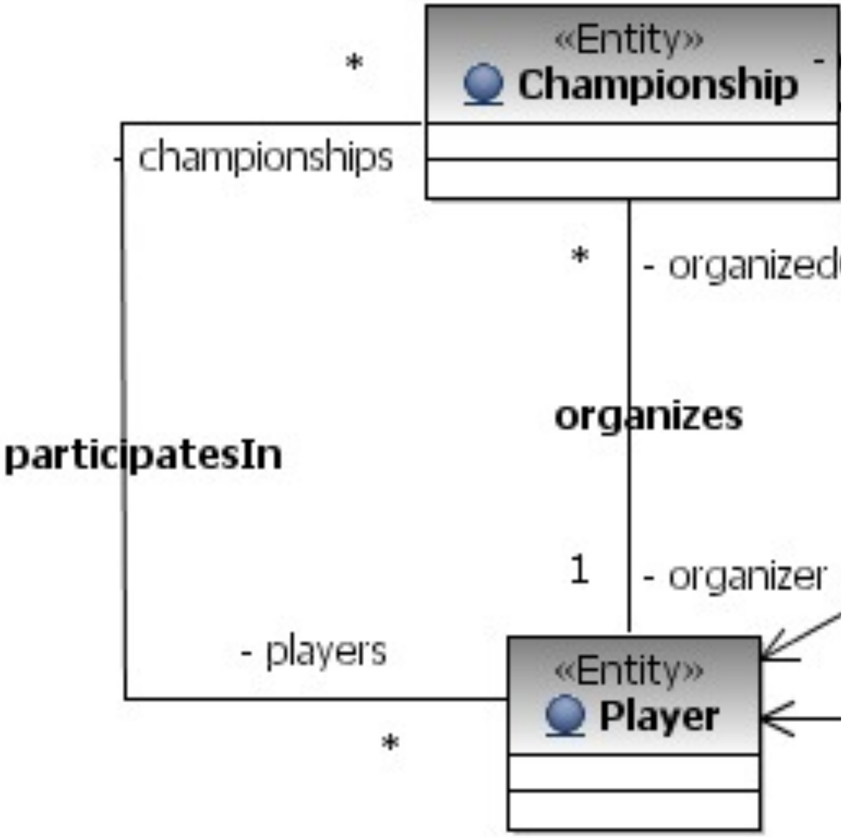


- A player is allowed to organize a single active championship at a time

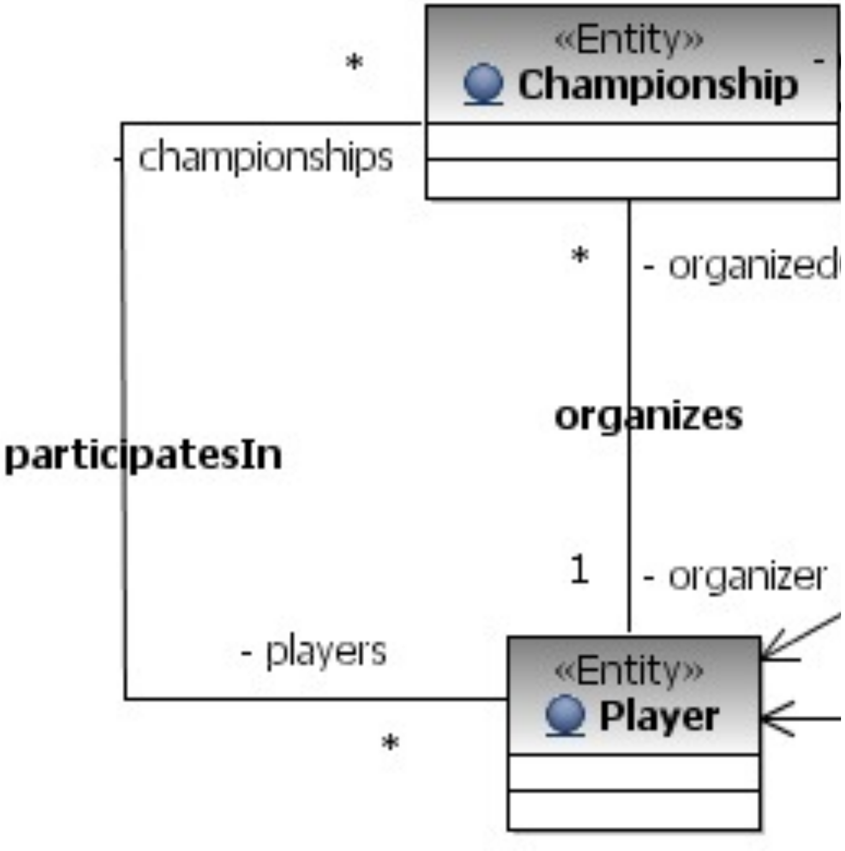
```
context Player inv:
    self.organized->forall(c1, c2 |
    c1<>c2 implies
    (c1.status = ChS::closed or
     c1.status = ChS::cancelled) or
    (c2.status = ChS::closed or
     c2.status = ChS::cancelled))
context Player inv:
    self.organized->select(c |
    c.status = ChS::announced or
```

Values of an enumeration

Application specific constraints

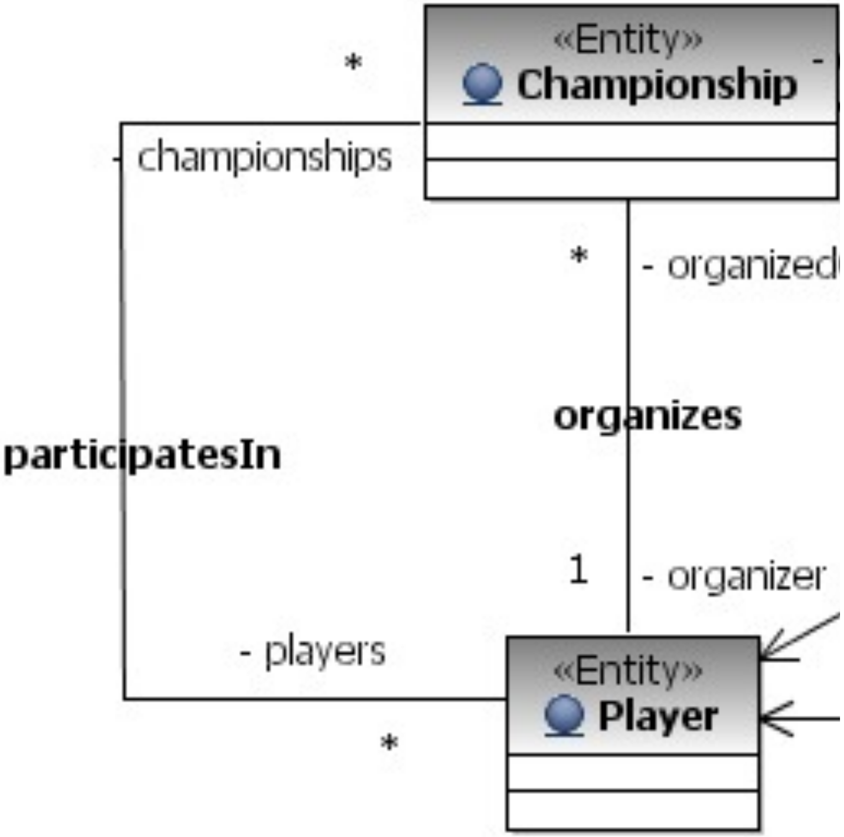


Application specific constraints



- A championship can only be started when the sufficient number of participants are present.

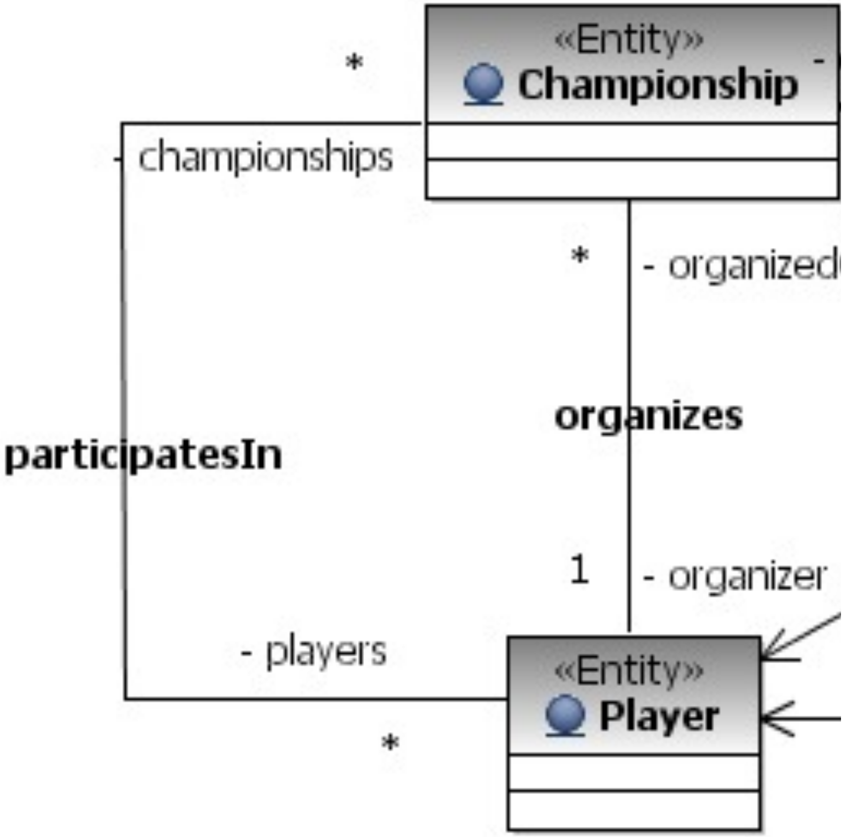
Application specific constraints



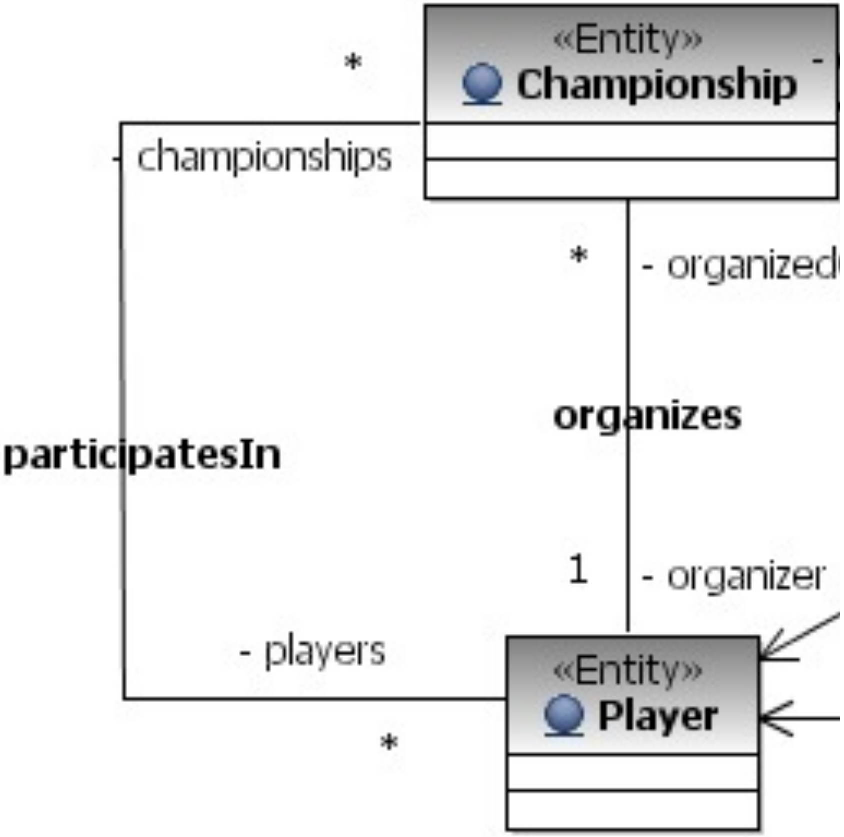
- A championship can only be started when the sufficient number of participants are present.

```
context Championship inv:
    self.status =
    ChampStatus::started or
    self.status =
    ChampStatus::finished
    implies
    (self.players->size >=
     self.minParticipants and
```

Application specific constraints

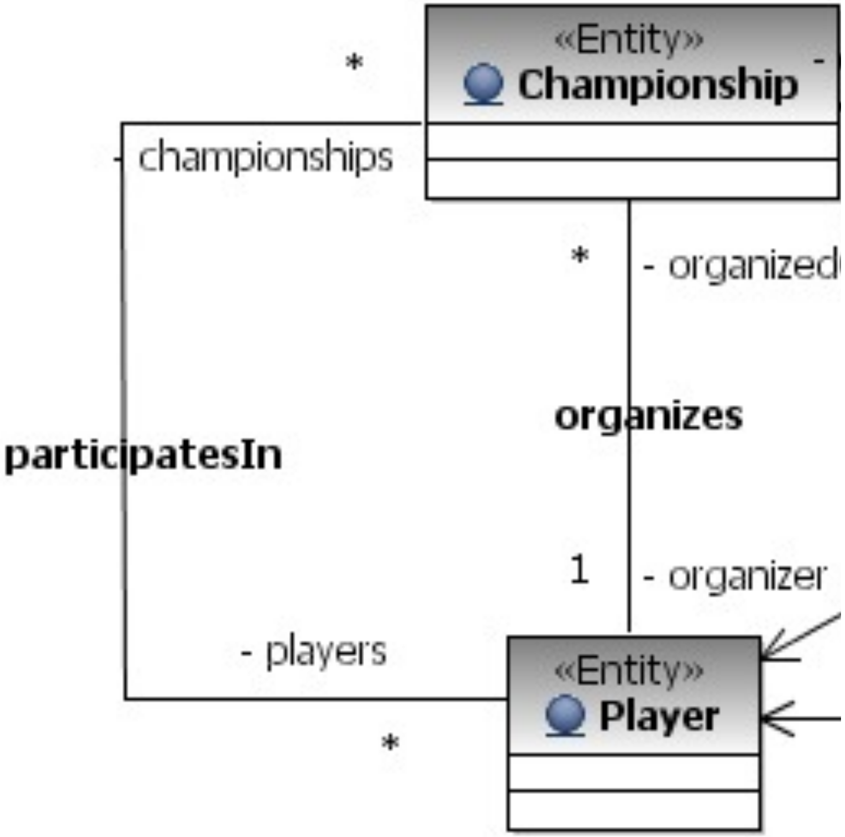


Application specific constraints



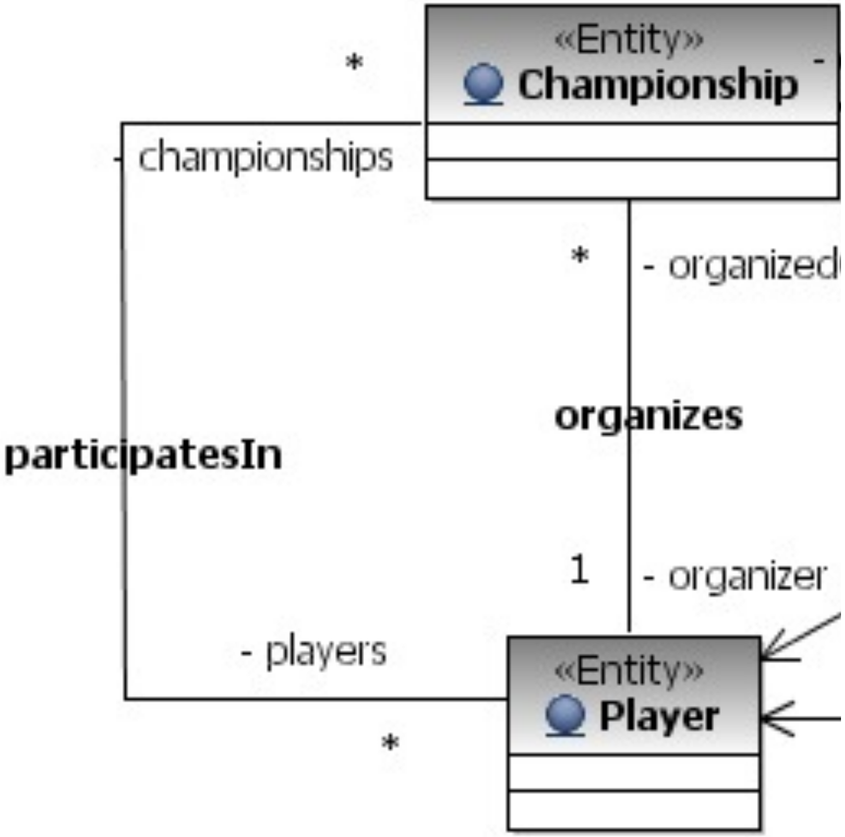
- Youth championship: the average age of participants is below 21.

Application specific constraints



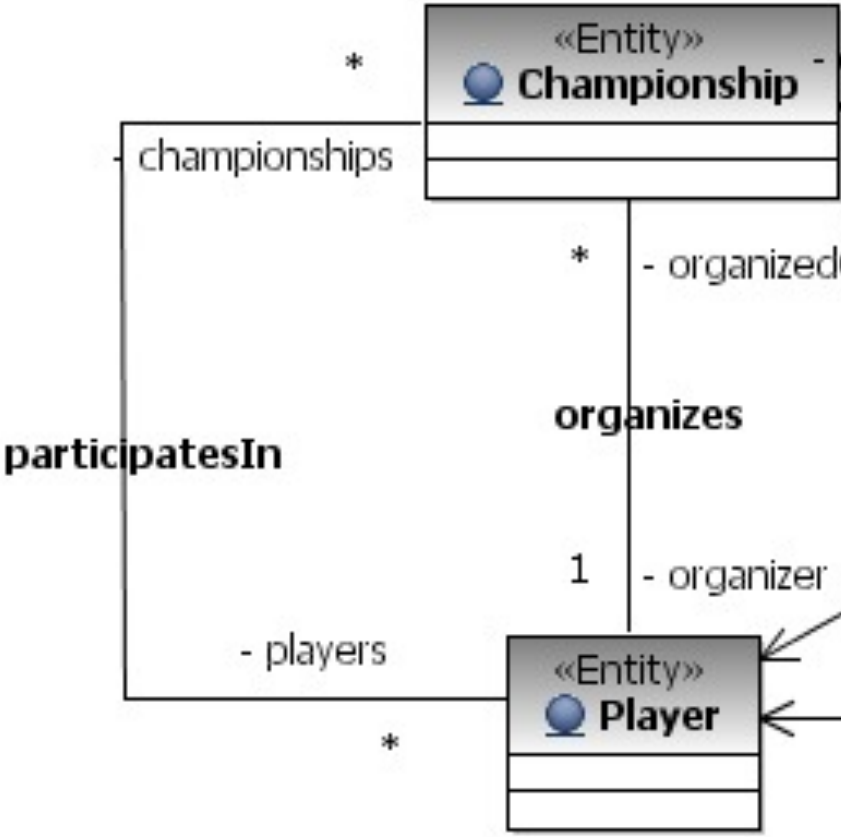
- Youth championship: the average age of participants is below 21.

Application specific constraints



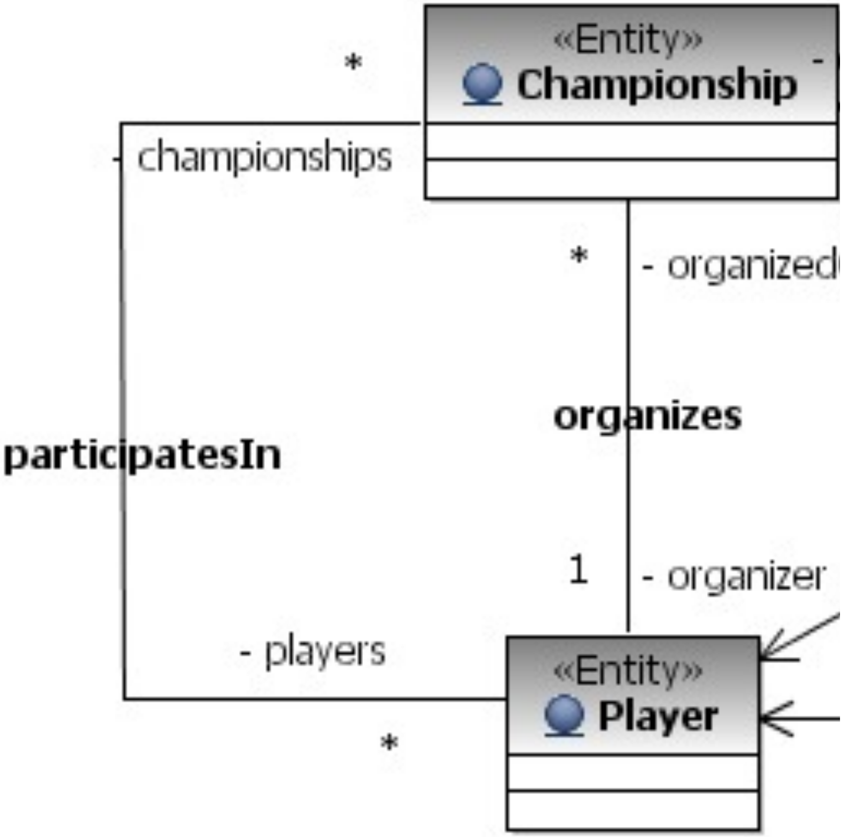
- Youth championship: the average age of participants is below 21.

Application specific constraints



- Youth championship: the average age of participants is below 21.

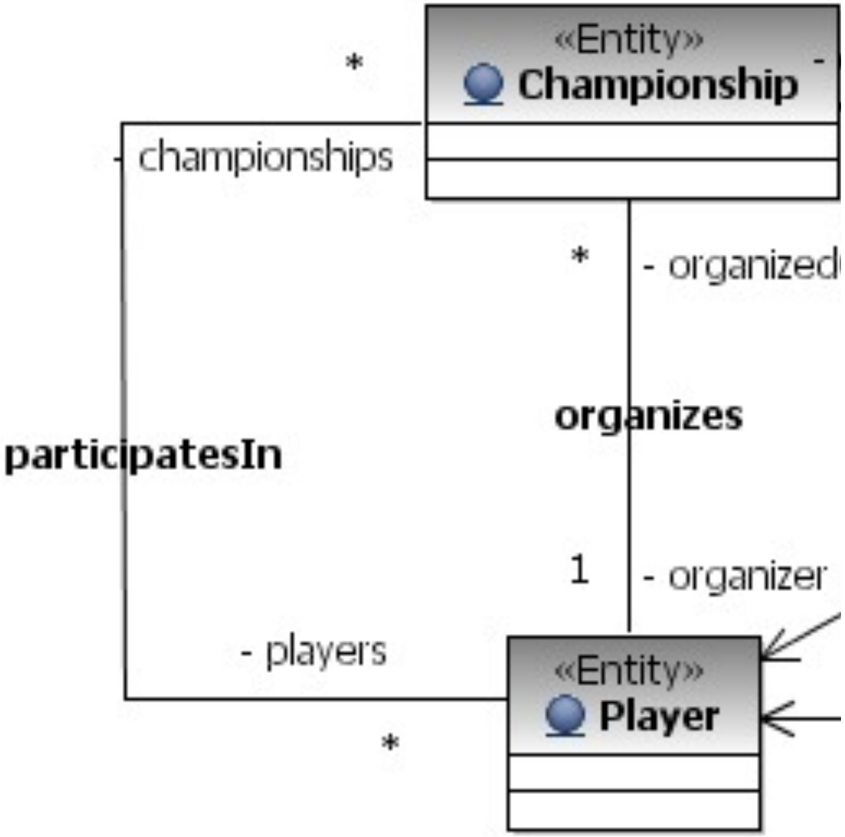
Application specific constraints



- Youth championship: the average age of participants is below 21.

```
context Championship inv:
   (self.players.age->sum) /
   (self.players->size) < 21
```

Application specific constraints



- Youth championship: the average age of participants is below 21.

players.age is the collection of the age attributes of players

```
context Championship inv:
  (self.players.age->sum) /
  (self.players->size) < 21
```

players.age->sum can only be applied to a collection that contains numbers

An Overview of OCL Constructs

Types and Boole algebra in OCL

- All OCL expressions are typed
  - OclAny:
    The type that includes all others. E.g. x, y : OclAny
  - x = y
    x and y are the same object.
  - x <> y
    not (x = y).
  - x.oclType
    The type of x.
  - x.isKindOf (T)
    True if T is a supertype (transitive) of the type of x.
  - T.allInstances : Collection
    All the instances of type T.

- Boolean operators:
  - b and b2, b or b2,
    b xor b2, not b
    If any part of a Boolean expression fully determines the result, then it does not matter if some other parts of that expression have unknown or undefined results.
  - b implies b2
    True if b is false or if b is true and b2 is true.
  - if b then e1 else e2 endif
    If b is true the result is the value of e1; otherwise, the result is the value of e2.

Overview of Collection Valued Terms

- Size:
  - c->size: Integer
    Number of elements in the collection; for a bag or sequence, duplicates are counted as separate items.
  - c->sum: Integer
    Sum of elements in the collection. Elements must be numbers
  - c->count(e): Integer
    The number of times that e is in c.
  - c->isEmpty: Boolean
    Same as (c->size = 0).
  - c->notEmpty: Boolean
    Same as (not c->isEmpty).
- Equality
  - c = c2 : Boolean
- Collection membership
  - c->includes(e): Boolean;
    c->exists (x | x = e).
  - c->excludes(e): Boolean;
    not c->includes(e).
  - c->includesAll(c2): Boolean;
    c includes all the elements in c2.
  - c->including(e): Collection
    The collection that includes all of c as well as e.
  - c->excluding(e): Collection
    The collection that includes all of c except e.

Overview of Collection Valued Terms

- Existential quantifier:
  - c->exists(x | P): Boolean; there is at least one element in c, named x, for which predicate P is true.
  - Equivalent notation is: c->exists(P), c->exists(x:Type | P(x))
- Universal quantifier:
  - c->forAll(x | P): Boolean; for every element in c, named x, predicate P is true.
  - Equivalent notation is: c->forAll(P) c->forAll(x:Type | P)
- Selection:
  - c->select(x | P): Collection The collection of elements in c for which P is true.
  - Equivalent is: c->select(P)
- Filtering:
  - c->reject(x | P): Collection c->select(x | not P).
  - Equivalent is: c->reject(P)
- Collection:
  - c->collect(x | E) : Bag The bag obtained by applying E to each element of c, named x.
  - c.attribute : Collection The collection(of type of c) consisting of the attribute of

Sets, Bags, Sequences

Definition:

Set{ 1, 2, 5, 88 }

Set{ 'apple', 'orange', 'strawberry'}

Sequence{ 1, 3, 45, 2, 3 }

Sequence{ 'ape', 'nut' }

Bag{1, 3, 4, 3, 5 }

Sequence{ 1..(5+4) } =

Sequence{ 1.. 9 } =

Sequence{ 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Set{ Set{1, 2}, Set{3, 4} }

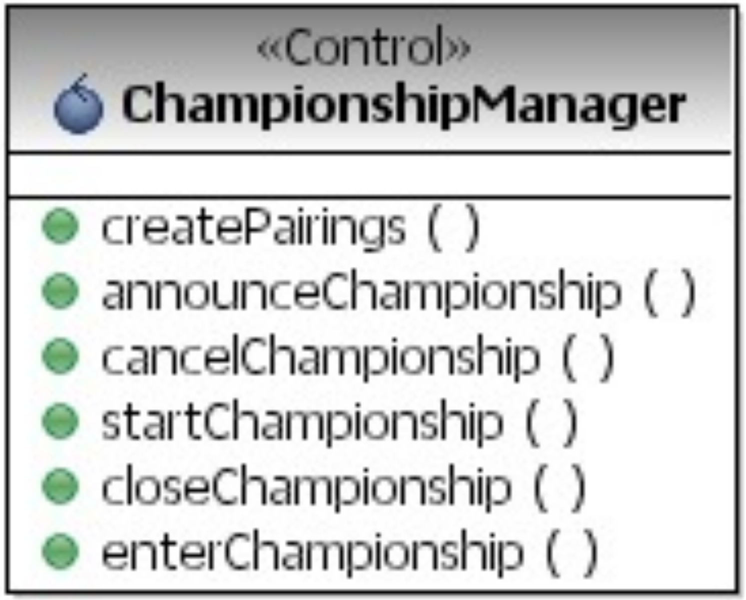
= Set{ 1, 2, 3, 4} (flattening)

Traditional operations are defined (union, intersection, etc.)

- Conversion from Collection:
  - c->asSet: Set
    A set corresponding to the collection (duplicates are dropped, sequencing is lost).
  - c->asSequence: Sequence
    A sequence corresponding to the collection.
  - c->asBag: Bag
    A bag corresponding to the collection.
- Comments:
  - --

Expressing Pre- and Postconditions of Operations

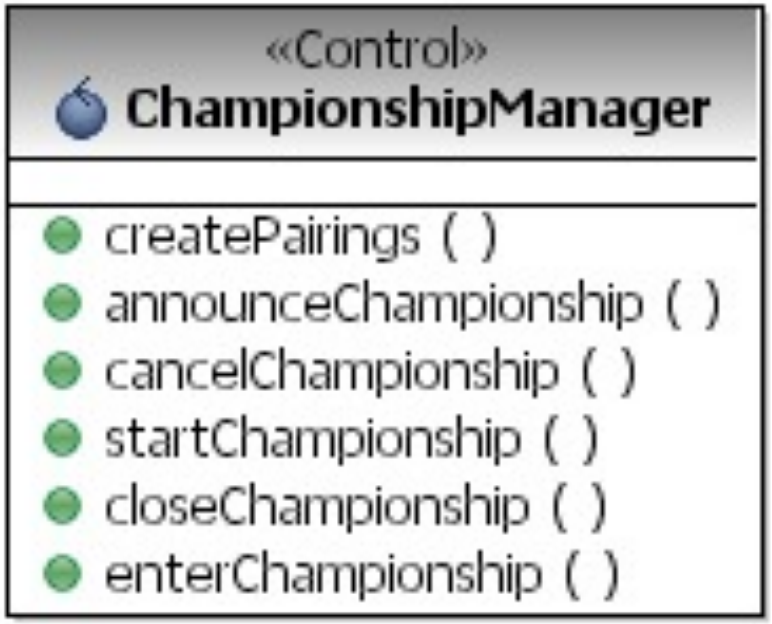
OCL Constraints of Operations



- Precondition: a condition that should hold before executing the operation
  - denoted by `pre:`

- Postcondition: a condition that should hold after executing the operation
  - denoted by `post:`

Constraints of Enter Championship



«Control»
ChampionshipManager

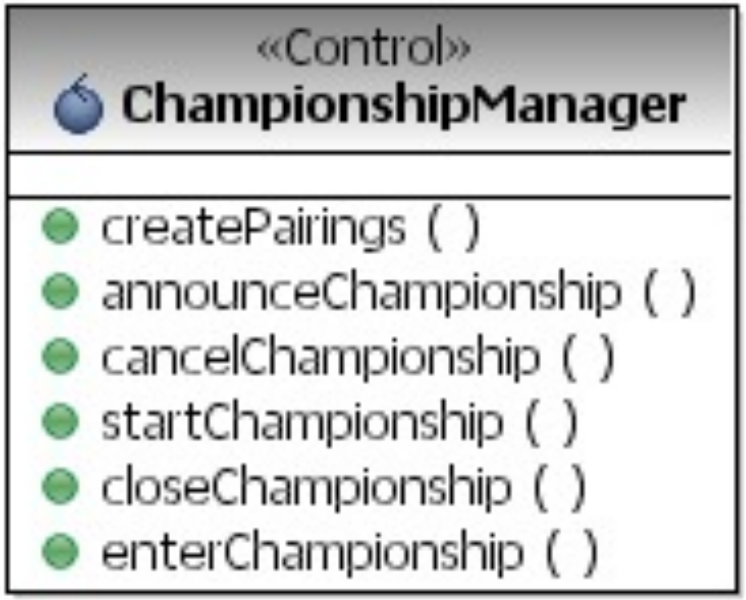
- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

- Signature

  ```
  void enterChampionship(
      Championship aChamp,
      Player aPlayer)
  ```

- Precondition
  - **aPlayer** is not yet a participant
  - **aChamp** is announced
- Postcondition
  - **aPlayer** becomes a participant

Constraints of Enter Championship



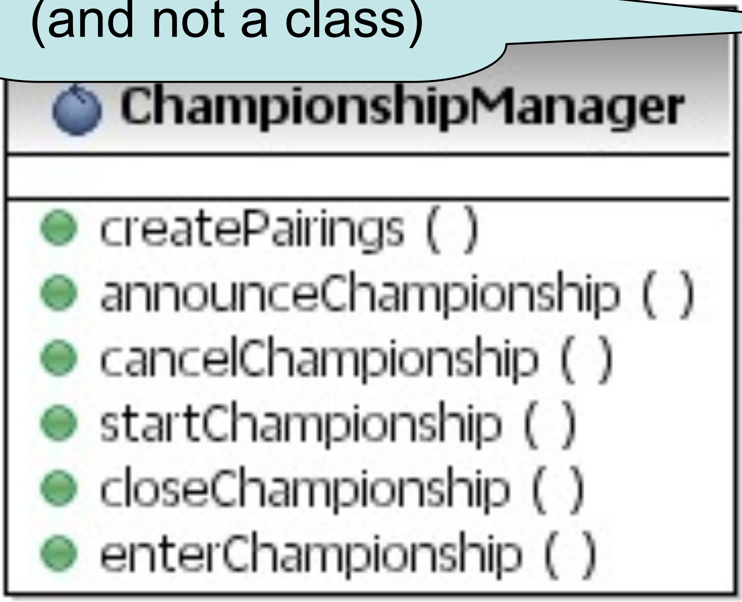
«Control»

ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

Constraints of
Enter Championship

The context is now
an operation
(and not a class)



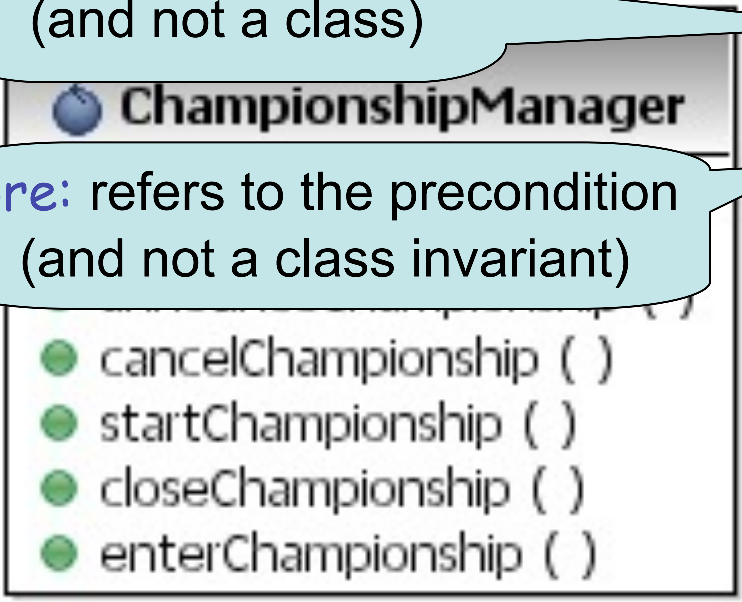
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
```

Constraints of Enter Championship

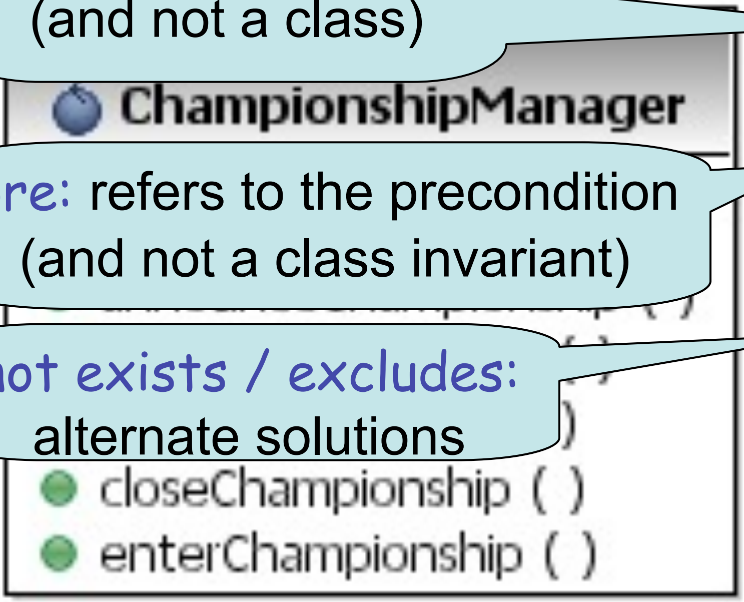
The context is now an operation (and not a class)



pre: refers to the precondition (and not a class invariant)

```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
pre:
    aPlayer.championships -> excludes
    (aChamp) and
```

Constraints of Enter Championship



The context is now an operation (and not a class)

pre: refers to the precondition (and not a class invariant)

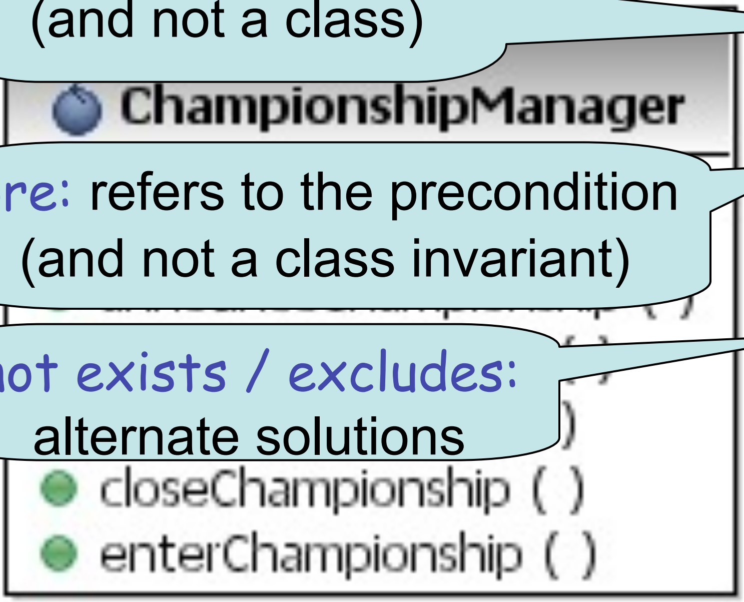
not exists / excludes: alternate solutions

```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
pre:
    aPlayer.championships -> excludes
    (aChamp) and
    not aChamp.players -> exists(p |
    p = aPlayer) and
    aChamp.status = ChS::announced
```

Constraints of Enter Championship

```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
pre:
    aPlayer.championships -> excludes
    (aChamp) and
    not aChamp.players -> exists(p |
    p = aPlayer) and
    aChamp.status = ChS::announced
post:
    aPlayer.championships =
    aPlayer.championships@pre ->
    including(aChamp) and
```

The context is now an operation (and not a class)



ChampionshipManager

closeChampionship ( )

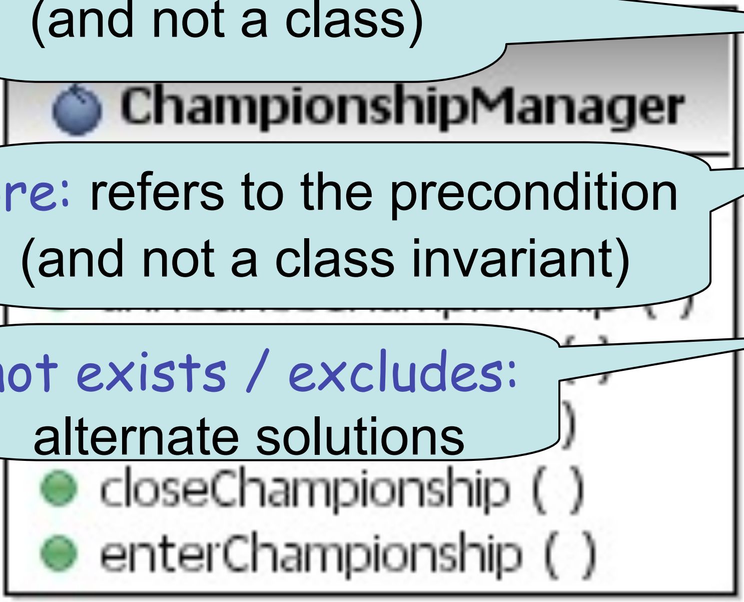
enterChampionship ( )

pre: refers to the precondition (and not a class invariant)

not exists / excludes: alternate solutions

Constraints of Enter Championship

The context is now an operation (and not a class)



```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
pre:
    aPlayer.championships -> excludes
    (aChamp) and
    not aChamp.players -> exists(p |
    p = aPlayer) and
    aChamp.status = ChS::announced
post:
    aPlayer.championships =
    aPlayer.championships@pre ->
    including(aChamp) and
    aChamp.players -> includes(aPlayer)
```

pre: refers to the precondition (and not a class invariant)

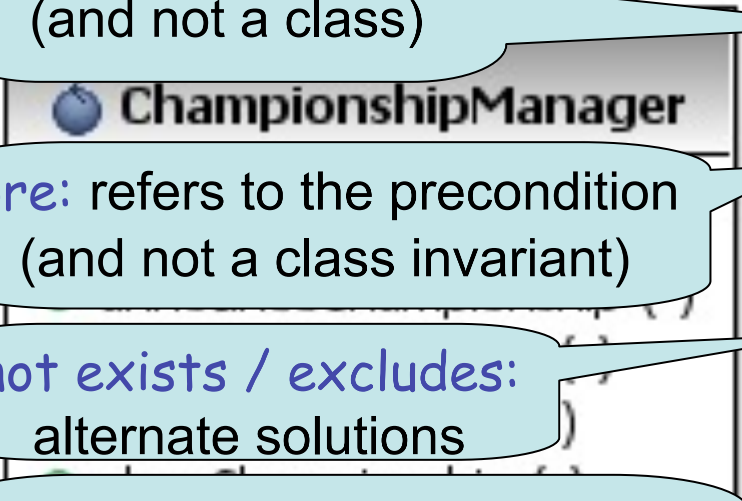
not exists / excludes: alternate solutions

Both roles of an assoc should be set

If omitted, the operation may change the status of a

Constraints of Enter Championship

```
context ChampionshipManager ::
    enterChampionship(
    Championship aChamp,
    Player aPlayer)
pre:
    aPlayer.championships -> excludes
    (aChamp) and
    not aChamp.players -> exists(p |
    p = aPlayer) and
    aChamp.status = ChS::announced
post:
    aPlayer.championships =
    aPlayer.championships@pre ->
    including(aChamp) and
    aChamp.players -> includes(aPlayer)
```



The context is now an operation (and not a class)

pre: refers to the precondition (and not a class invariant)

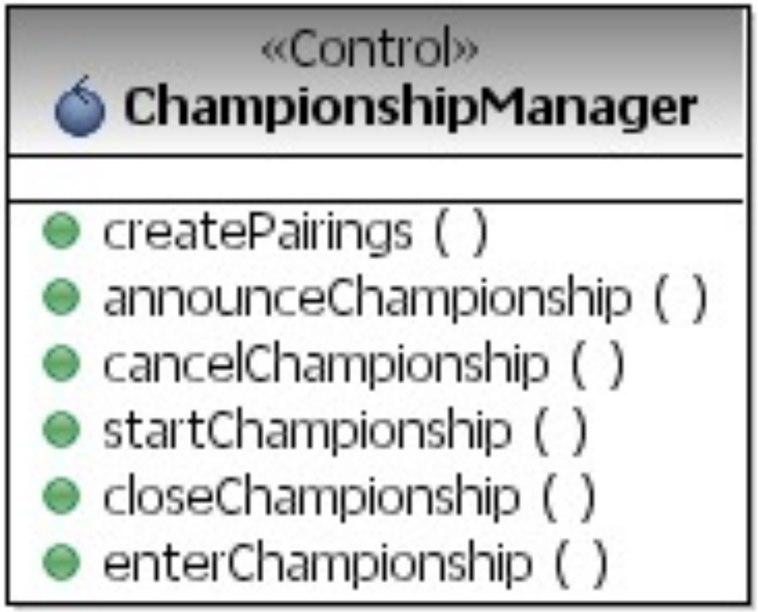
not exists / excludes: alternate solutions

@pre refers to the value of a term before the operation is executed

Both roles of an assoc should be set

If omitted, the operation may change the status of a

Constraints of Announce Championship



«Control»
ChampionshipManager

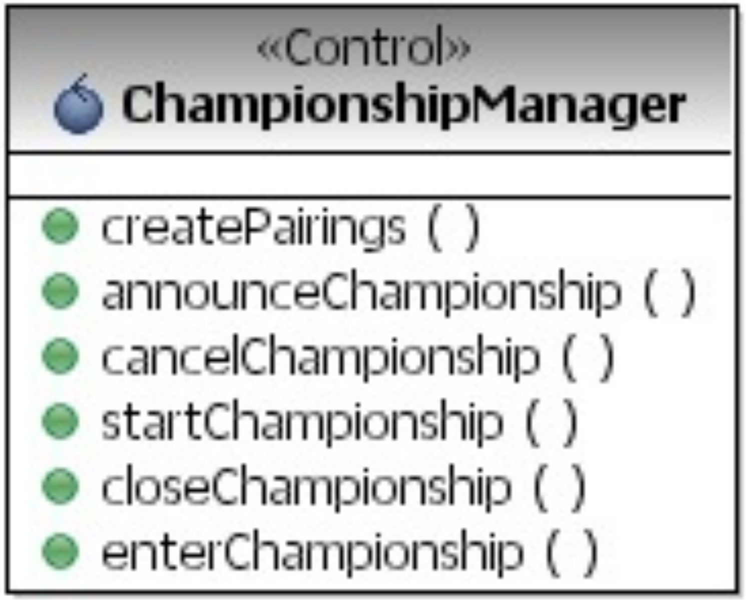
createPairings ( )
announceChampionship ( )
cancelChampionship ( )
startChampionship ( )
closeChampionship ( )
enterChampionship ( )

- Signature

```
Championship announceChampionship(
    String aName,
    Player anOrganizer,
    Integer aMinParticipant,
    Integer aMaxParticipant)
```

- Precondition:
  - Min and max values are between bounds
  - Organizer does not have active champs
- Postcondition:
  - The collection of championship instances includes a new one with

Constraints of Announce Championship

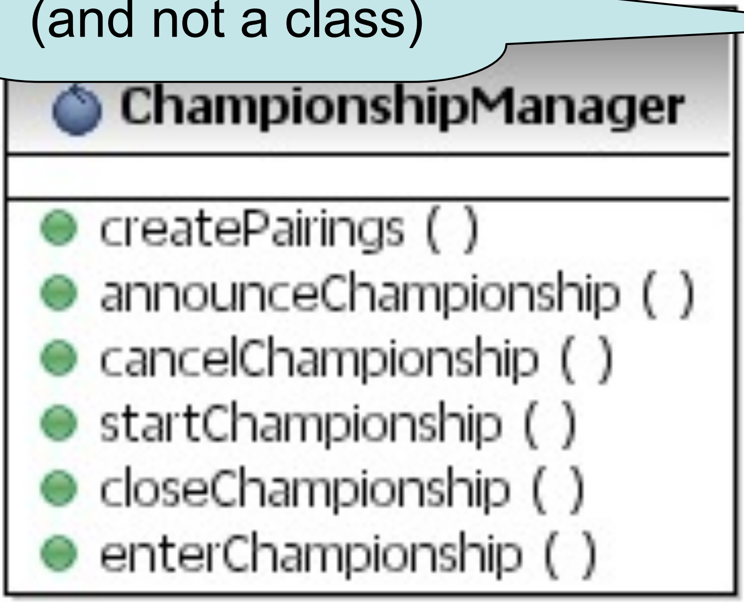


«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

Constraints of Announce Championship

The context is now an operation (and not a class)



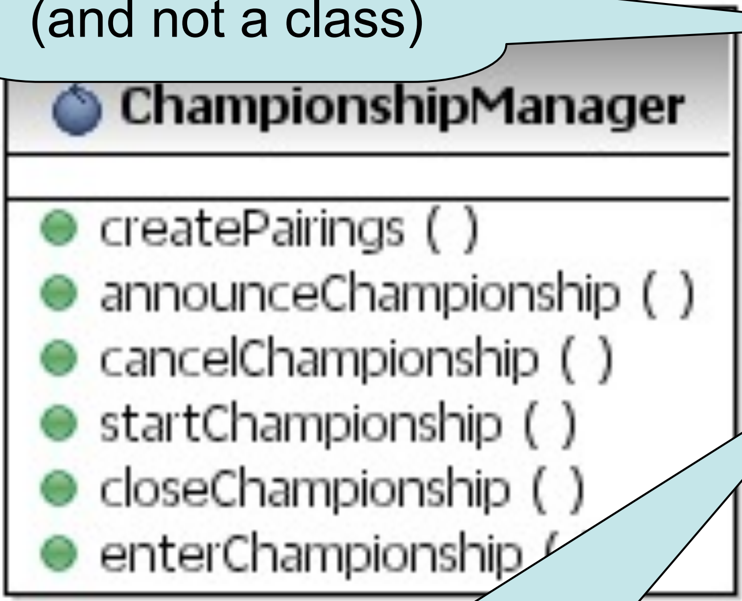
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    announceChampsionship( String
    aName,
    Player anOrganizer,
    Integer aMinParticipant,
    Integer aMaxParticipant)
```

Constraints of Announce Championship

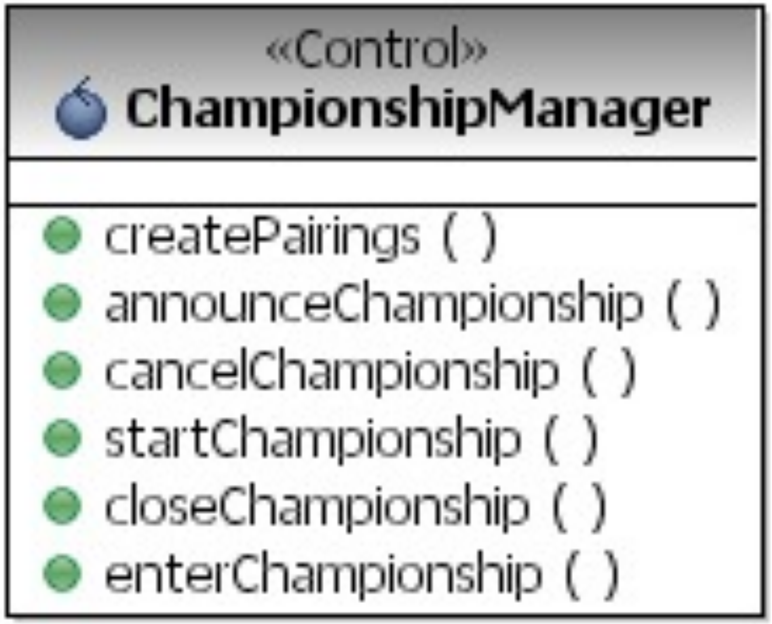
The context is now an operation (and not a class)



pre: refers to the precondition (and not a class invariant)

```
context ChampionshipManager ::
    announceChampsionship( String
    aName,
    Player anOrganizer,
    Integer aMinParticipant,
    Integer aMaxParticipant)
pre:
    (aMinParticipant >= 0 and
     aMaxParticipant > 0 and
     aMinParticipant <= aMaxParticipant)
    and
    anOrganizer.organized->forall( c |
    c.status = ChS::cancelled or
    c.status = ChS::closed)
```

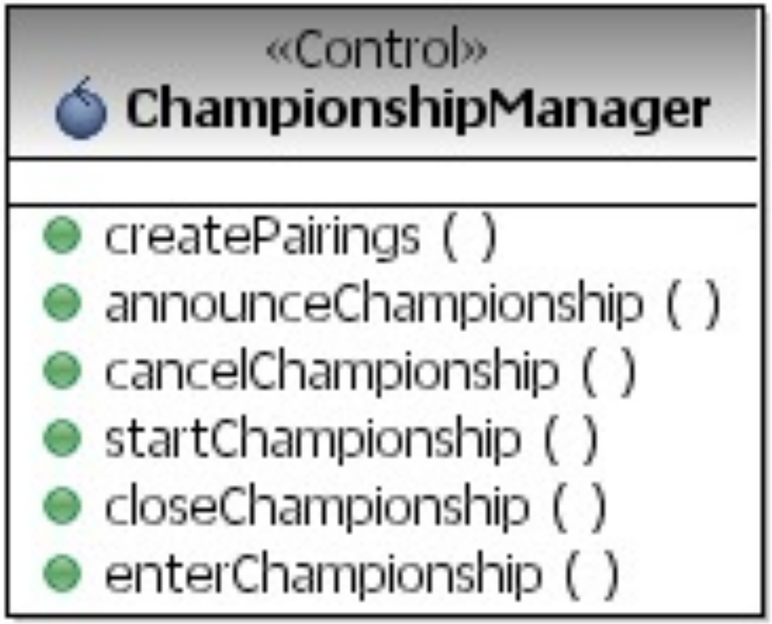
Constraints of Announce Championship



«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

Constraints of Announce Championship

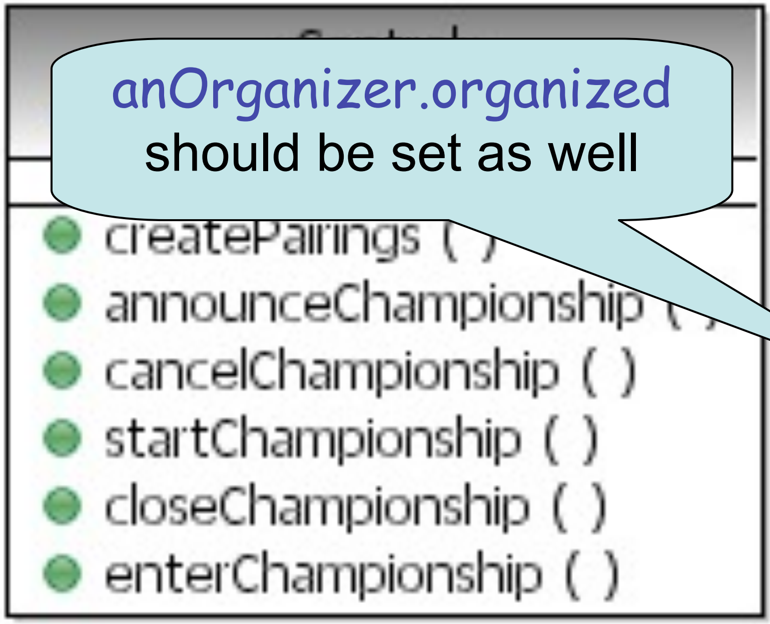


«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
post: -- Solution 1
    Championship.allInstances ->
    exists(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
    c.organizer = anOrganizer
```

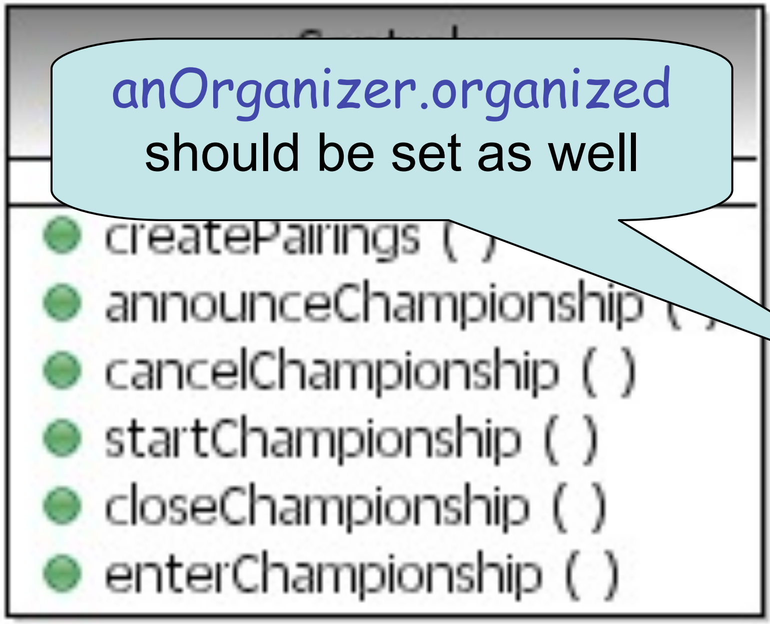
Constraints of Announce Championship



anOrganizer.organized should be set as well

```
post: -- Solution 1
    Championship.allInstances ->
    exists(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
    c.organizer = anOrganizer
    and
    anOrganizer.organized -> includes(c))
```

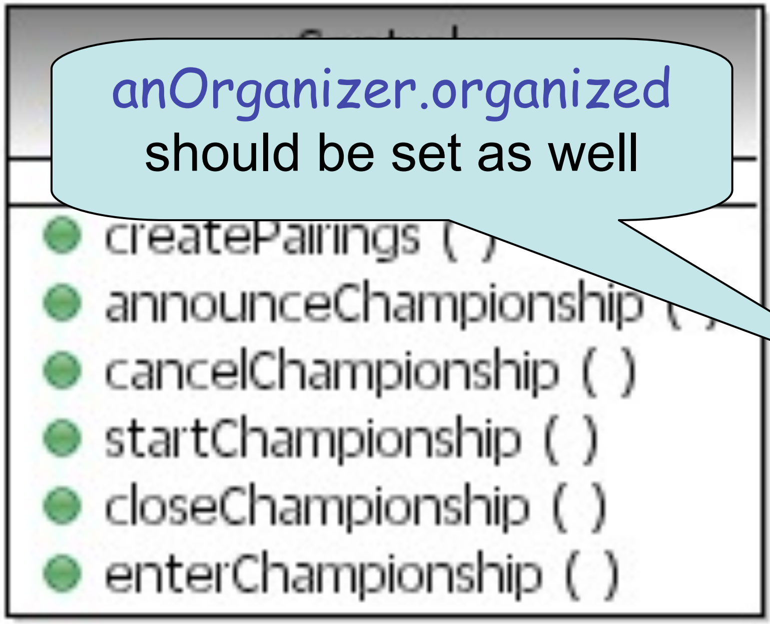
Constraints of Announce Championship



anOrganizer.organized should be set as well

```
post: -- Solution 1
    Championship.allInstances ->
    exists(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
    c.organizer = anOrganizer
    and
    anOrganizer.organized -> includes(c))
```

Constraints of Announce Championship

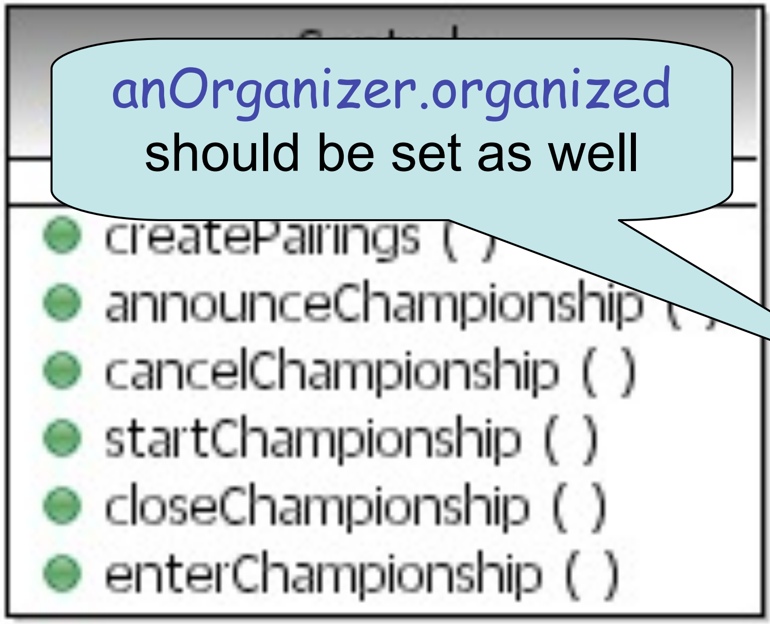


anOrganizer.organized should be set as well

```
post: -- Solution 1
    Championship.allInstances ->
    exists(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
    c.organizer = anOrganizer
    and
    anOrganizer.organized -> includes(c))

post: -- Solution 2
    Championship.allInstances =
    Championship.allInstances@pre->
    including(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
```

Constraints of Announce Championship



anOrganizer.organized should be set as well

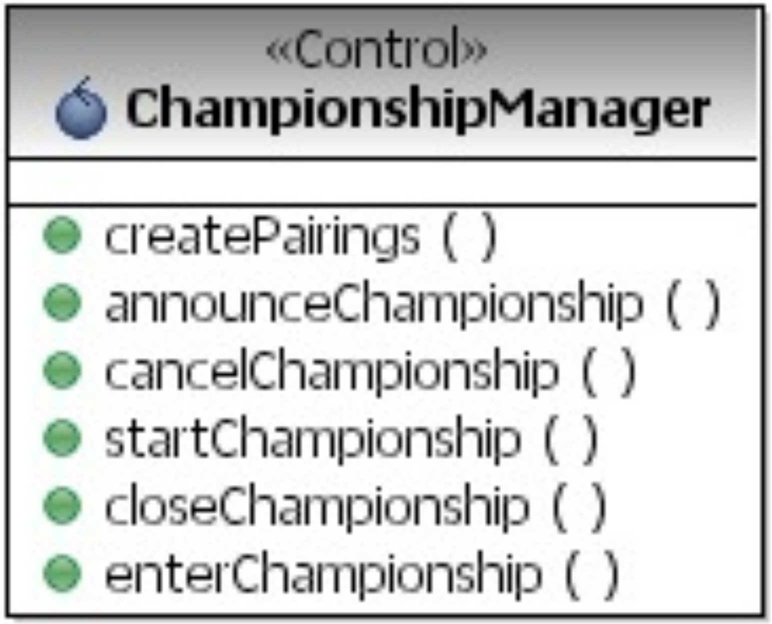
```
post: -- Solution 1
    Championship.allInstances ->
    exists(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
    c.organizer = anOrganizer

    and
    anOrganizer.organized -> includes(c))
```

```
post: -- Solution 2
    Championship.allInstances =
    Championship.allInstances@pre->
    including(c | c.name = aName and
    c.minParticipant = aMinParticipant and
    c.maxParticipant = aMaxParticipant and
```

@pre refers to the value of a term before the operation is executed

Constraints of Start Championship



«Control»
ChampionshipManager

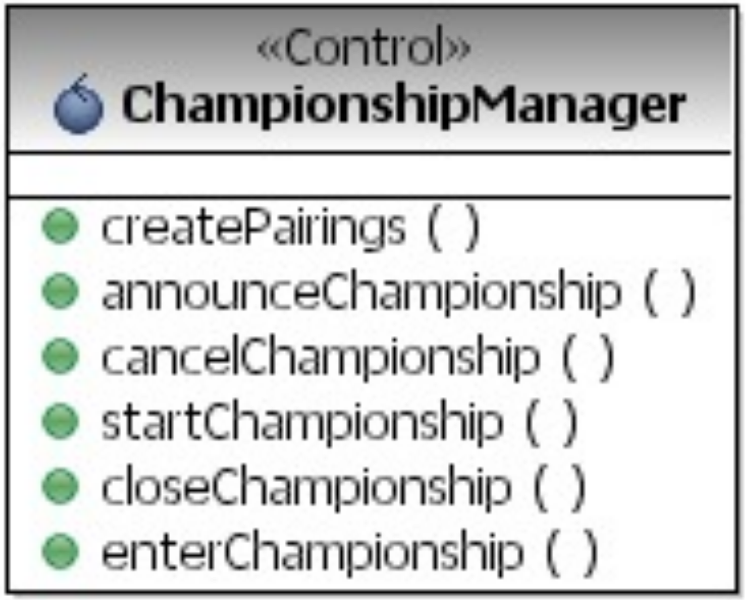
createPairings ( )
announceChampionship ( )
cancelChampionship ( )
startChampionship ( )
closeChampionship ( )
enterChampionship ( )

- Signature

  void startChampionship(
  Championship aChamp)

- Precondition
  - ***aChamp*** is announced
  - the number of participants is between limits
- Postcondition
  - ***aChamp*** is started

Constraints of Start Championship

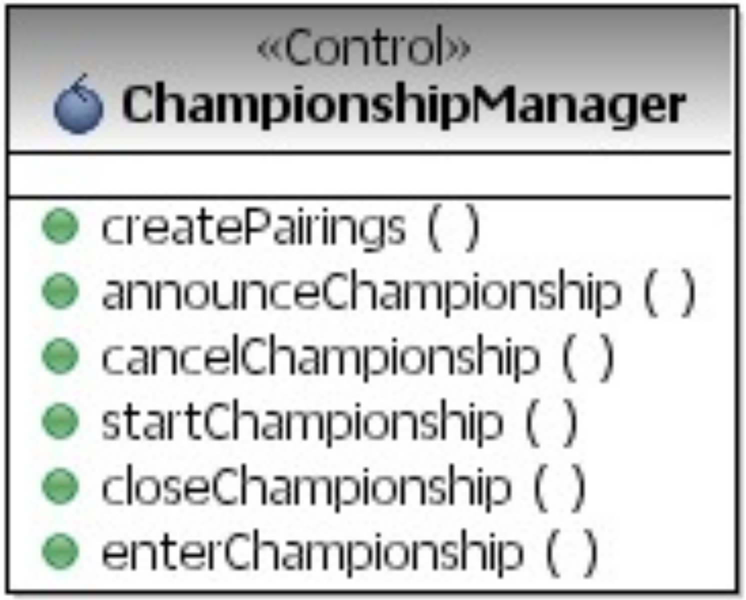


«Control»

ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

Constraints of Start Championship

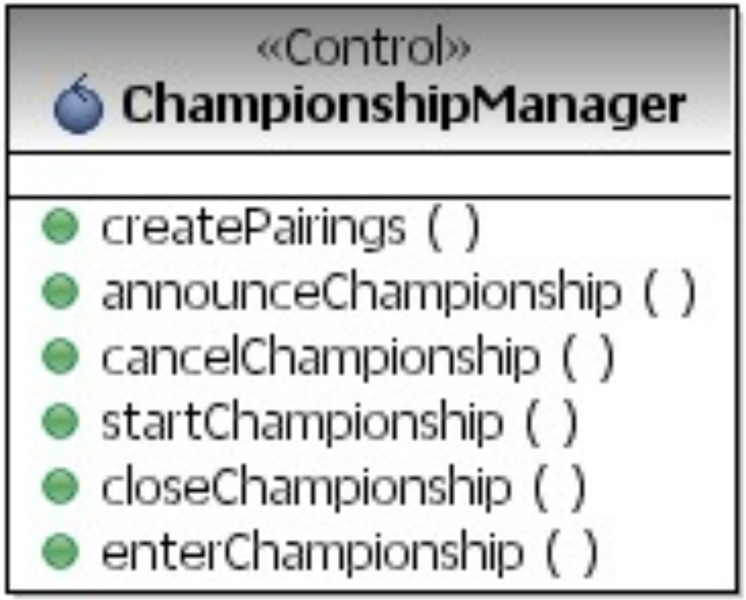


«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    startChampsionship(
    Championship aChamp)
```

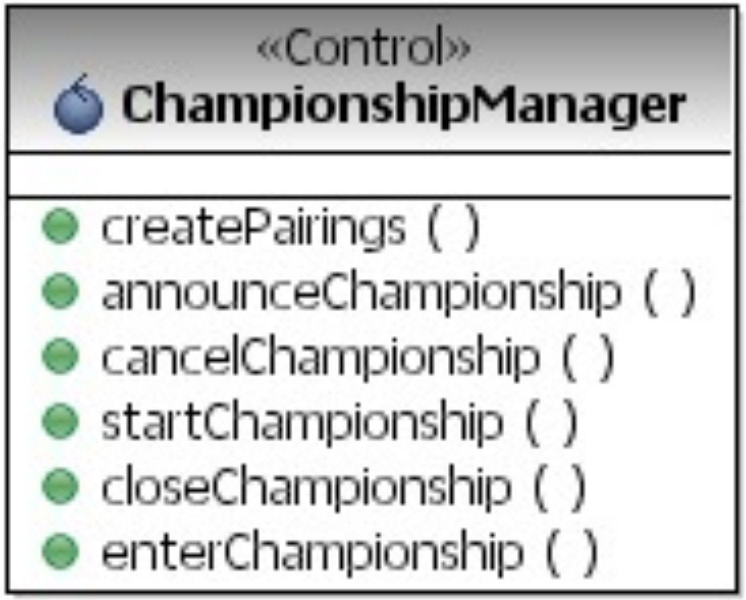
Constraints of Start Championship



```
context ChampionshipManager ::
    startChampsionship(
    Championship aChamp)

pre:
    aChamp.status = ChS::announced
    aChamp.players -> size >=
    aChamp.minParticipant and
    aChamp.players -> size <=
    aChamp.maxParticipant
```

Constraints of Start Championship



«Control»
ChampionshipManager

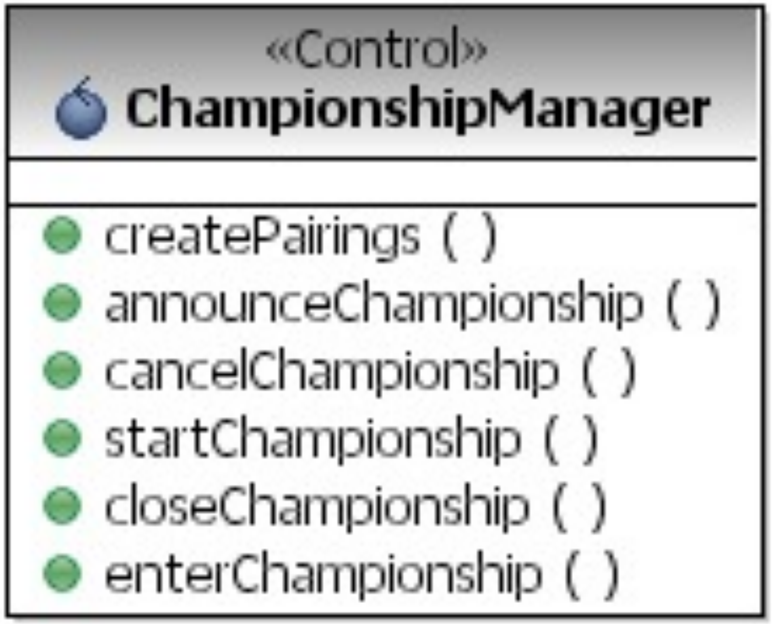
- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    startChampsionship(
    Championship aChamp)

pre:
    aChamp.status = ChS::announced
    aChamp.players -> size >=
    aChamp.minParticipant and
    aChamp.players -> size <=
    aChamp.maxParticipant

post:
    aChamp.status = ChS::started
```

Constraints of Cancel Championship



«Control»
ChampionshipManager

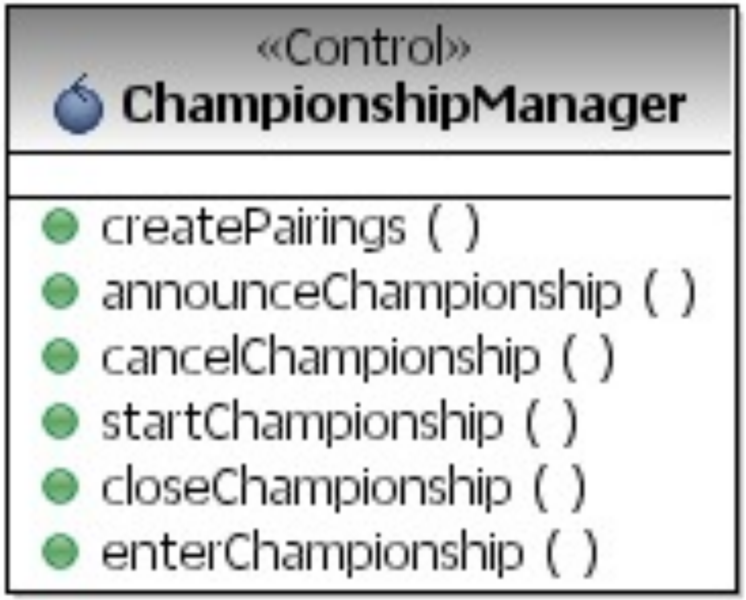
createPairings ( )
announceChampionship ( )
cancelChampionship ( )
startChampionship ( )
closeChampionship ( )
enterChampionship ( )

- Signature

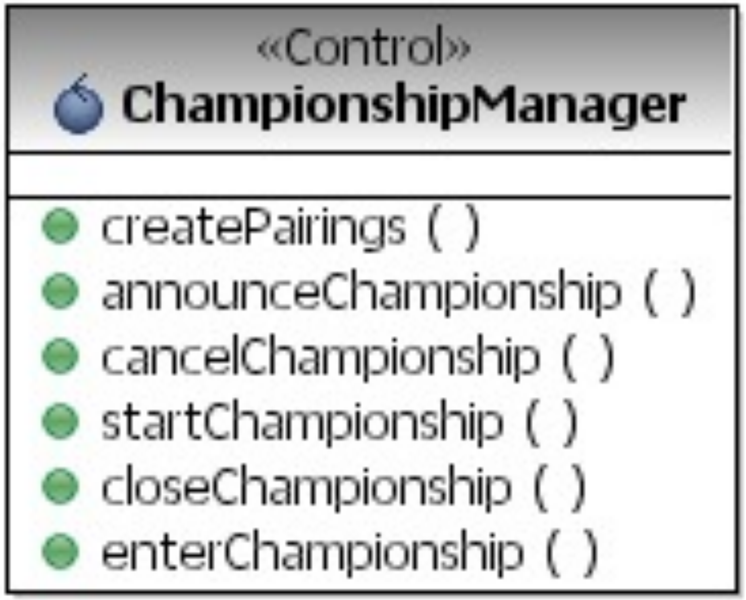
  void cancelChampionship(
    Championship aChamp)

- Precondition
  - **aChamp** is announced
- Postcondition
  - **aChamp** is cancelled

Constraints of Cancel Championship



Constraints of Cancel Championship



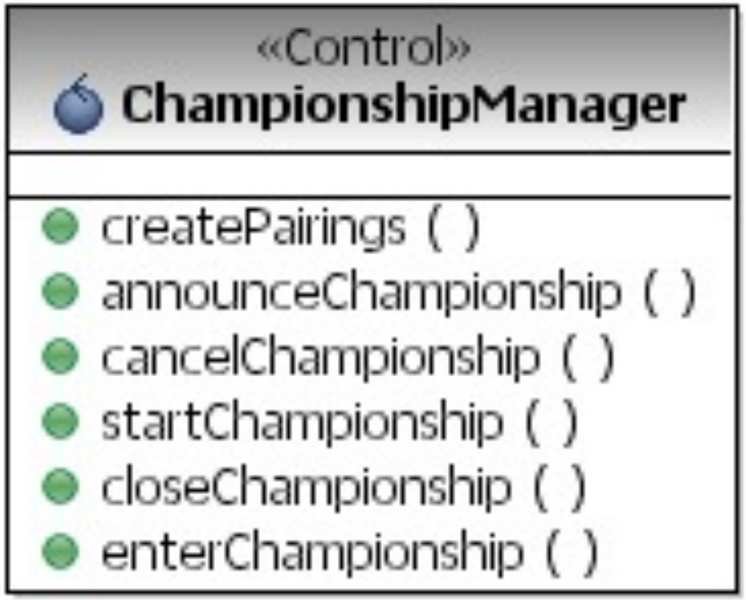
«Control»

ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    cancelChampsionship(
    Championship aChamp)
```

Constraints of Cancel Championship



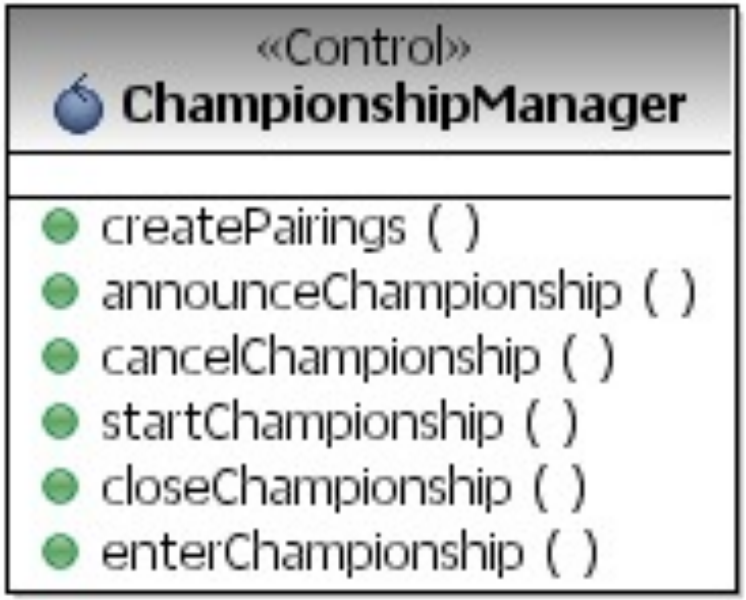
«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    cancelChampsionship(
    Championship aChamp)

pre:
    aChamp.status = ChS::announced
```

Constraints of Cancel Championship



«Control»
ChampionshipManager

- createPairings ( )
- announceChampionship ( )
- cancelChampionship ( )
- startChampionship ( )
- closeChampionship ( )
- enterChampionship ( )

```
context ChampionshipManager ::
    cancelChampsionship(
    Championship aChamp)
pre:
    aChamp.status = ChS::announced
post:
    aChamp.status = ChS::cancelled
```

What restrictions cannot be captured in OCL?

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship

Temporal constraints!!!

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship

Temporal constraints!!!

G (not (started B cancel))

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship

Temporal constraints!!!

~~G (not (started B cancel))~~

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - ~~If the championship is not started yet (e.g. the number of participants does not reach a minimum level), the organizer may cancel the championship~~

Temporal constraints!!!

~~G (not (started B cancel))~~

Verbal Requirements

- Requirements:
  - A player should register and log in before using the system
  - Each registered player may announce a championship.
  - Each player is allowed to organize a single championship at a time.
  - Players may join (enter) a championship on a web page
  - When the sufficient number of participants are present, the organizer starts the championship.
  - After starting a championship, the system must automatically create the pairings in a round-robin system.
  - The organizer may cancel the championship ONLY IF the championship is not started yet

G (started -> F (not(cancel)))

Next Lecture: Architecture Modeling

- How to integrate existing components?
- Typical architectures of web applications

Questions

- Can a single object act as a set?
  - E.g. c.organizer.size
- Referring to constraints
- Return values?
- If sg is not changed by an operation, should we state it explicitly?