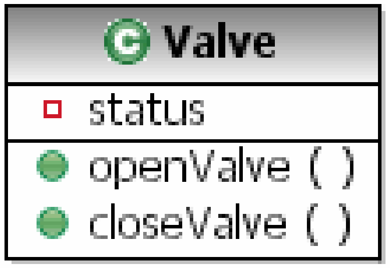
Structural Design in UML with Analysis Classes

UML based modeling and analysis

Dániel Varró

Traditional OO Design

- A Class encapsulates
  - Attributes of the class (instance)
  - Operations performed on the class (instance)
- Appropriate for embedded systems where
  - Classes are strongly related to real objects of the system (e.g. Valve)
  - Operations are strongly related a single class E.g. *openValve()*
  - Operations correspond to real operations E.g. *openValve()* opens a real valve
- The tradition OO view turned out to be problematic (especially in web applications)



Problems of OO Modeling in Web Applications

- Where to put business functionality?
  - a) `champ.enterChampionship(Player p)`
  - b) `player.enterChampionship(Championship c)`
- Proposal:

```
ChampionshipManagement mngr;
mngr.enterChampionship(Championship c, Player p)
```

- Essence of the proposal:
  - – Encapsulate business functionality into a separate interface (class): `ChampionshipManager`
  - – Make persistent business data reusable: `Player`

Problems of OO Modeling in Web Applications

- Where to put GUI handler code?
  - a) `championship.enterButtonClicked(Event e)`
  - b) `manager.enterButtonClicked(Event e)`
- Proposal:

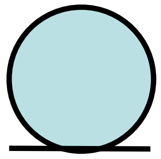
```
PlayerEnterChampForm form;
form.enterButtonClicked(Event e)
```

- Essence of the proposal:
  - – Encapsulate user interfaces into separate classes: `PlayerEnterChampForm`
  - – Keep business functionality separated from GUI handlers

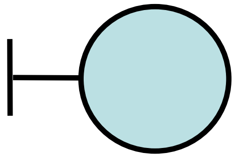
How to Structure the Structure or How to classify classes?

Analysis Classes

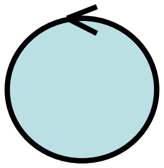
- Entity class (Entitás osztály):
  - Persistent data
    (used multiple times and in many UCs)
  - Still exists after the UC terminates (e.g. DB storage)



- Boundary class (Határoló osztály):
  - (User) interface between actors and the system
  - E.g. a Form, a Window (Pane)



- Control class (Vezérlő osztály):
  - Encapsulates business functionality



- Proposed in RUP (Rational Unified Process)

Rules of Thumb for Analysis Classes

Structural restrictions for analysis classes

- Entity: only attributes (+get/set/find methods)
- Control: only methods: (at least) one method / UC
- Boundary: both attributes and methods

Relationship between analysis classes (Layers)

- Actors access only boundaries
- One boundary class for each Actor-UC relation
- Entities are only accessed by control objects
- Control objects may communicate with all entities, boundaries, and control objects

Example:
Championship Manager

Verbal Requirements

- Design a system for organizing championships of table games (chess, go, backgammon, etc.)
- Requirements:
  - A player should register and log in to the system before using it.
  - 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.

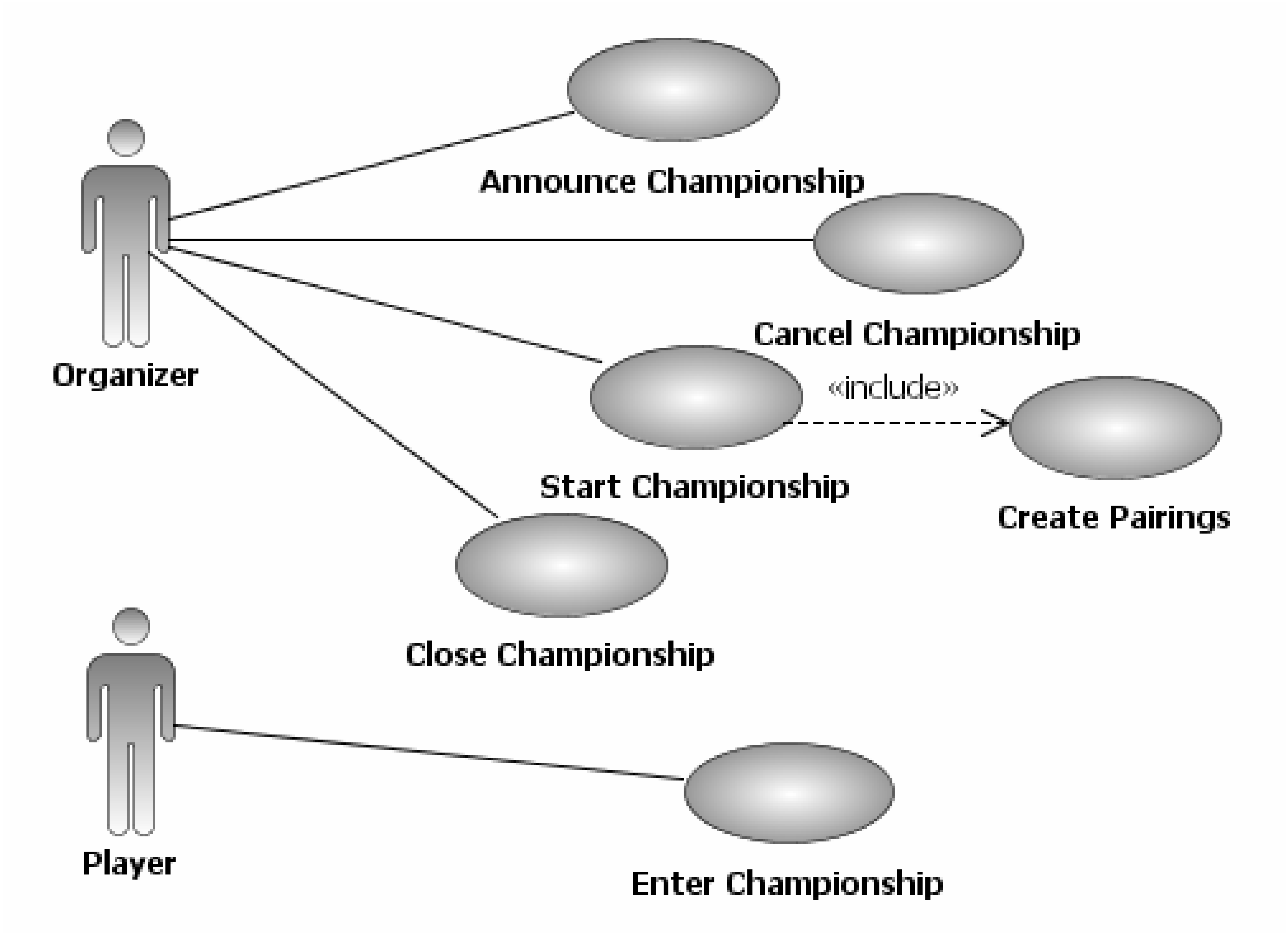
Verbal Requirements (cont.)

- Requirements (cont.):
  - 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
  - The actual game is played between existing clients, which is outside the scope of the system system.
  - Both players should report the result and the moves after each game using a web form. A win scores 1 point, a draw ½, and a loss 0.
  - If players report contradicting results, the organizer should judge who is the winner. The organizers penalizes the cheating player by a 1 point penalty.
  - When all games are finished, the organizer should close the championship by announcing the winner. Then he or she may start organizing a new championship.

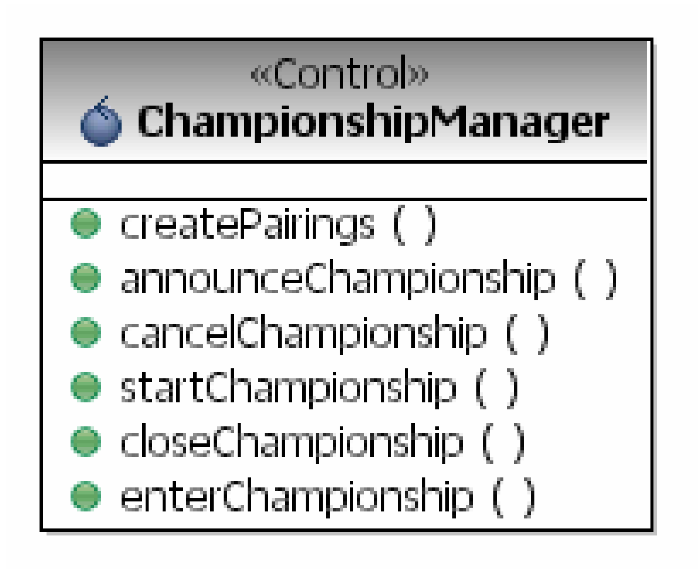
Requirements (cont.)

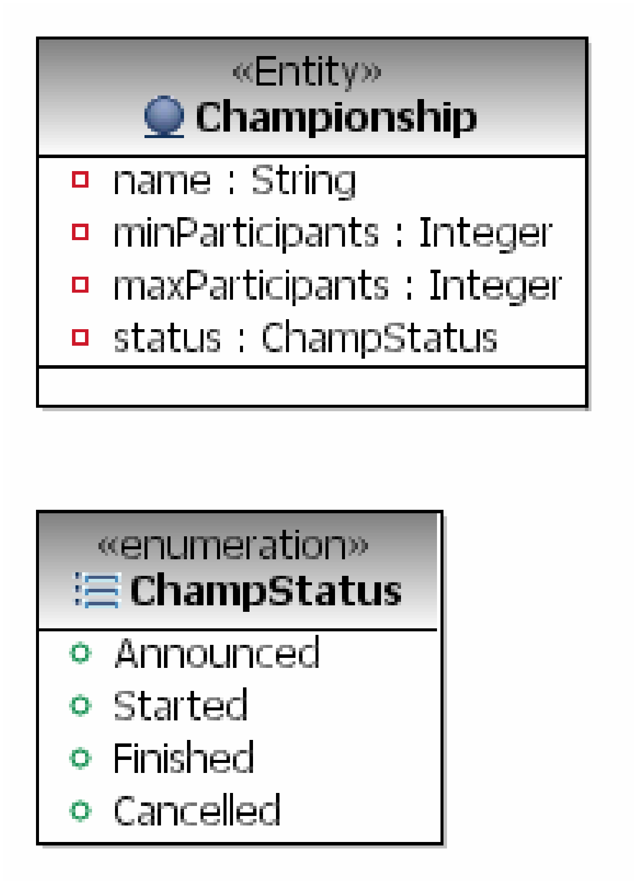
- A game should be finished within a given deadline (time limit).
- If none of the two players have reported the result within this deadline, then both players are considered to be losers.
- If only one player has reported the result, then his (or her) version is considered to be the official result.

Championship Management



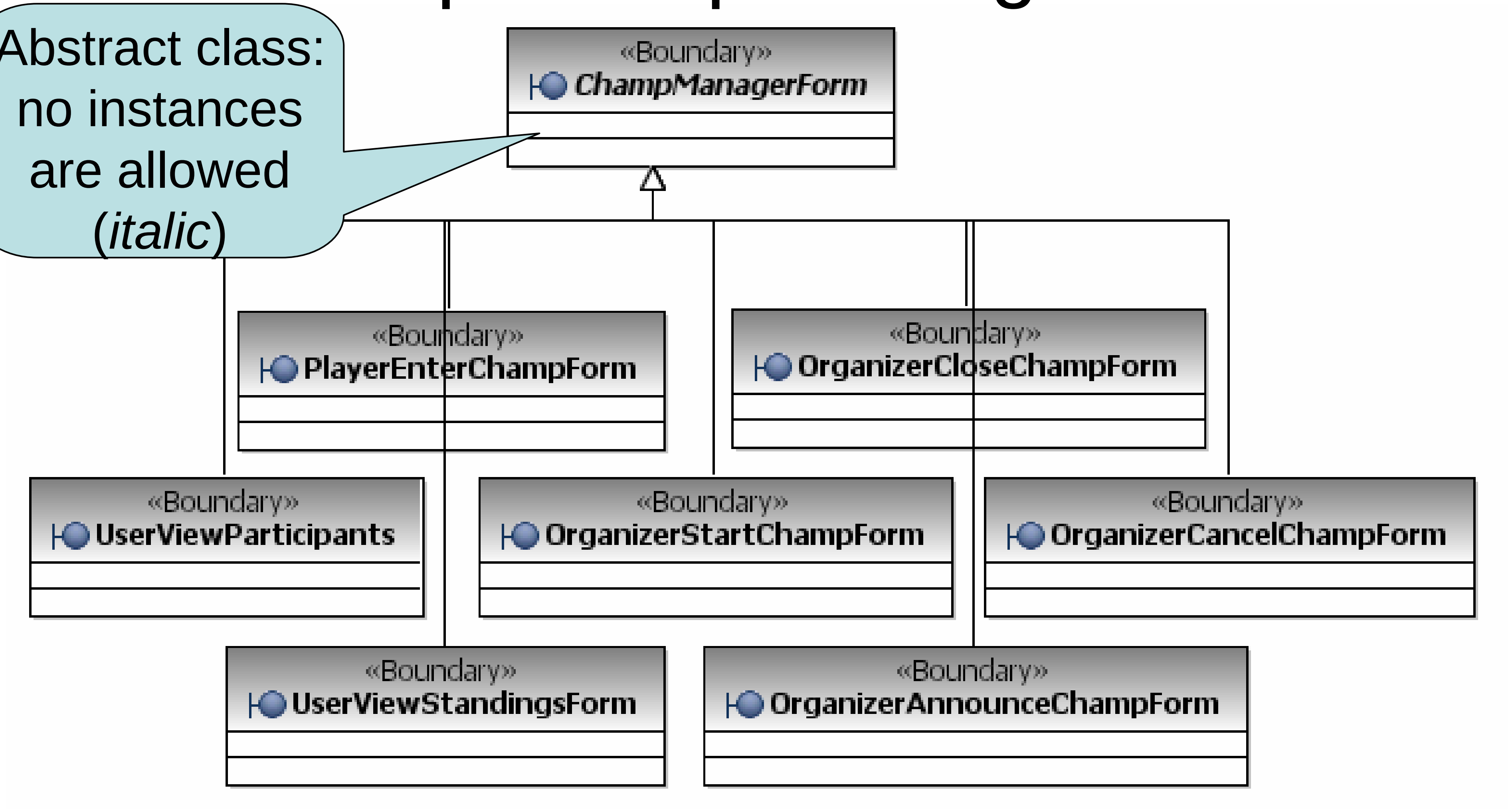
Control and Entity Classes for Championship Management





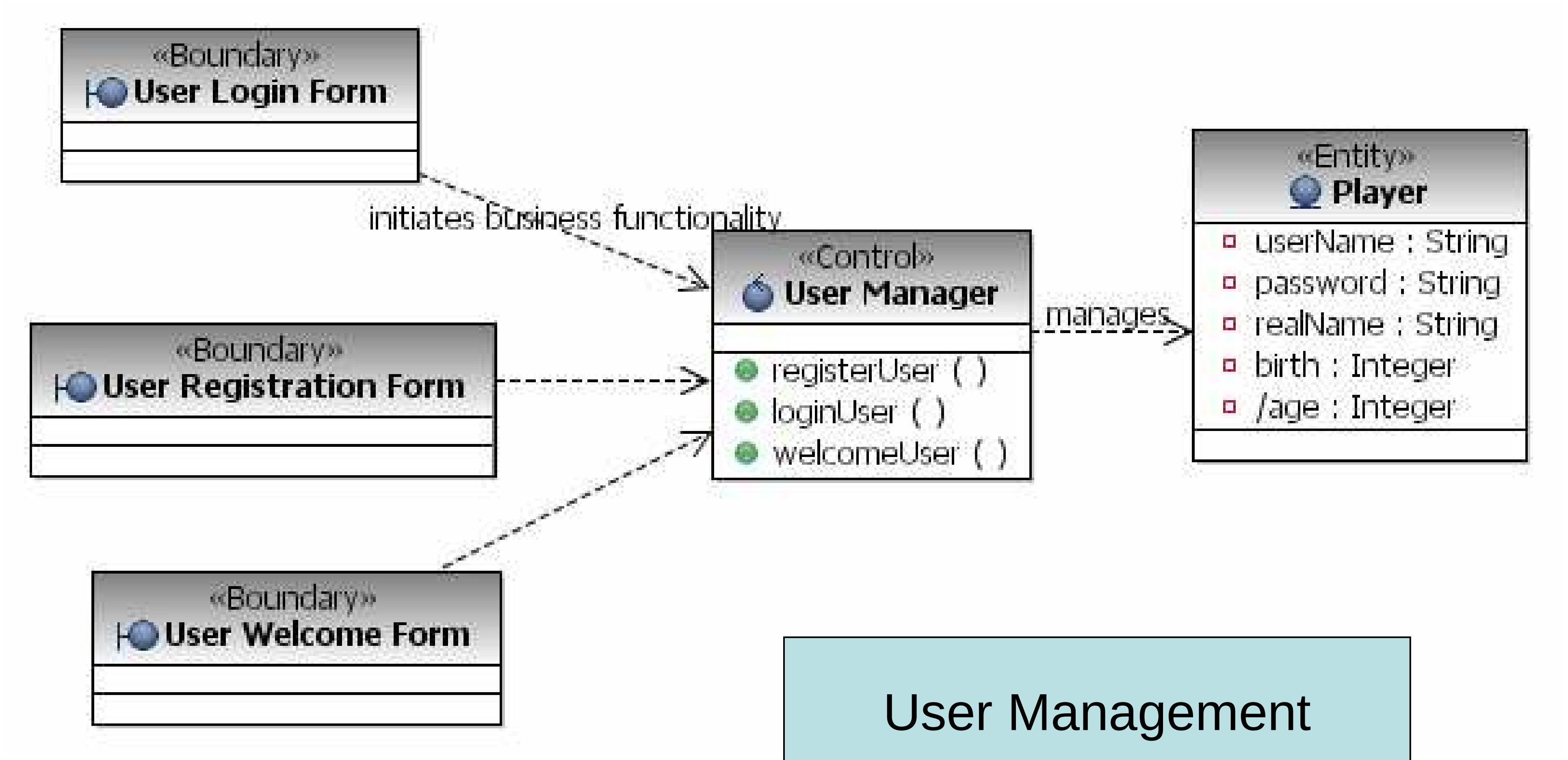
Boundary Classes for Championship Management

Abstract class: no instances are allowed (*italic*)



Detailed design of boundary classes will come later

Relationship between Analysis Classes



User Management

Organization of Analysis Models

- Analysis Model
  - Championship Management Package
    - Analysis Elements Package
      - Entity classes
      - Control classes
      - Boundary classes
      - Enumerations
      - Subpackages
    - Collaborations (Not discussed today)
  - Game Management Package
  - User Management Package

Syntactic Best Practice of Class Diagrams

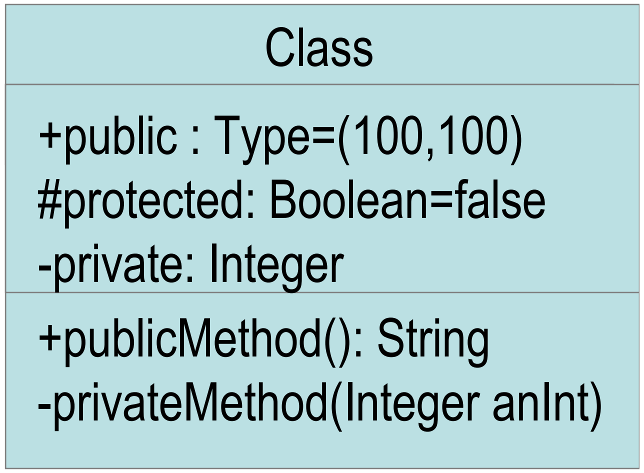
- Limit the number of classes in a single diagram. Divide large diagrams into smaller ones
- Naming:
  - Class: domain-specific noun
  - Operations: with a strong action verb
  - Attributes: descriptive noun
- Level of details
  - Analysis-level vs. Design-level
  - Do not mix them!
- Preferrable arrangement of relations
  - Associations: horizontal
  - Generalizations: vertical

Structure Modeling with
Entity Classes and Associations

Traditional Classes

Class

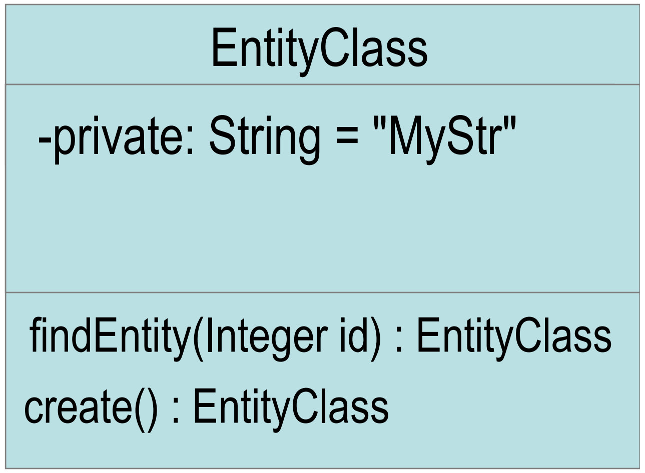
- name
- attributes (attribútumok)
  - Visibility (láthatóság)
  - Type (típus)
  - Initial value (kezdőérték)
- methods (metódusok)
  - Visibility (láthatóság)
  - Type (típus)
  - Query vs. Manipulation



Entity Classes

Entity Class

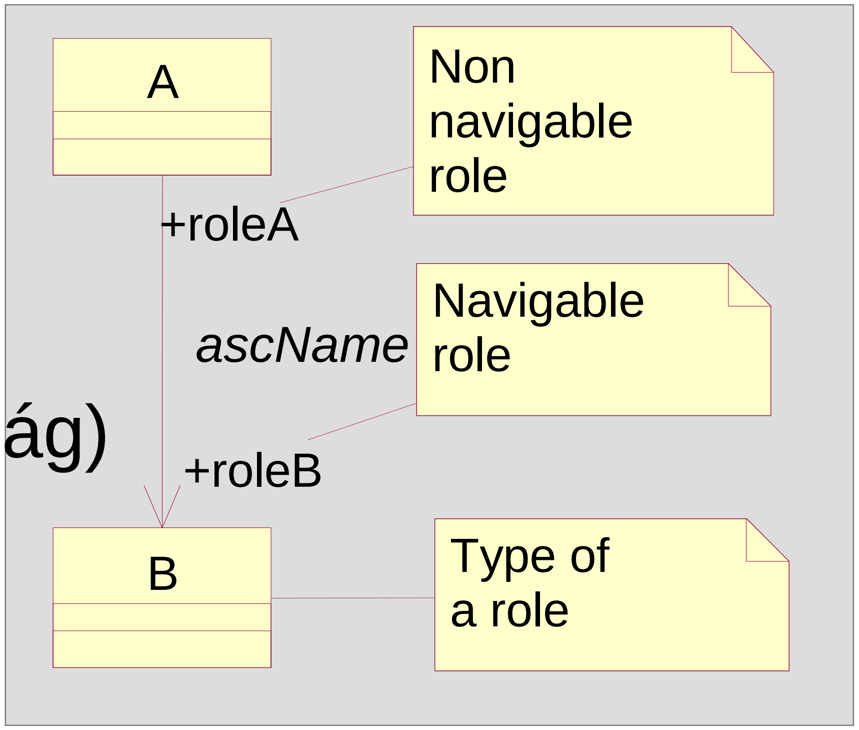
- name
- attributes (attribútumok)
  - Visibility: private / irrelevant
  - Type: important
  - Initial value: rarely relevant
- methods (metódusok)
  - Only Find and Create in the analysis model
  - Only Get/Set in the design model


EntityClass
-private: String = "MyStr"
findEntity(Integer id) : EntityClass
create() : EntityClass

Associations between Entity Classes

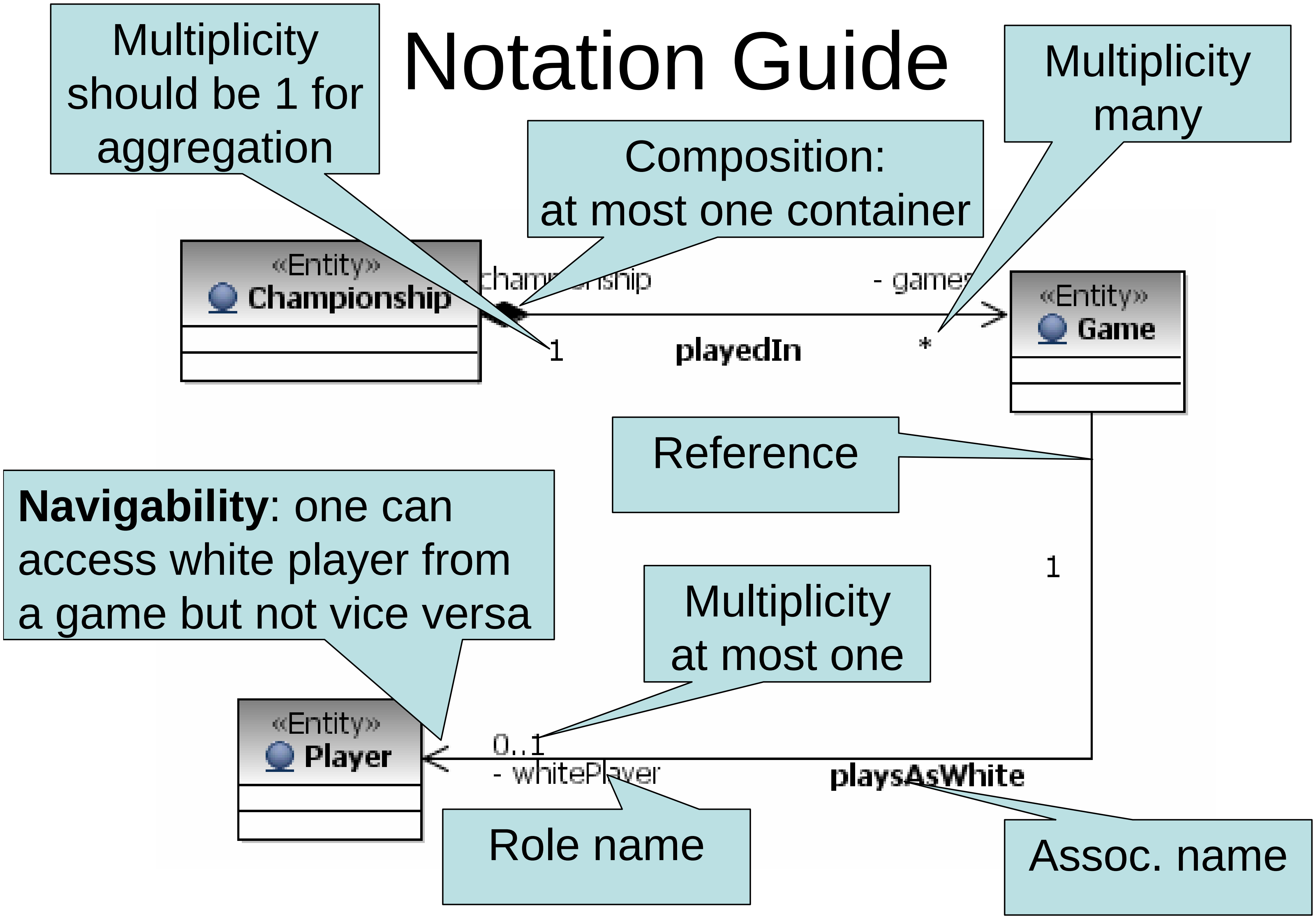
Association (Asszociáció):
relationship between (objects of) classes

- Name (név)
- Role (szerep)
  (for each Assoc. End)
  - Role name (szerep név)
  - Navigability (navigálhatóság)
  - Multiplicity (multiplicitás)
  - Type (típus)



- Composition (Aggregation) vs. Reference

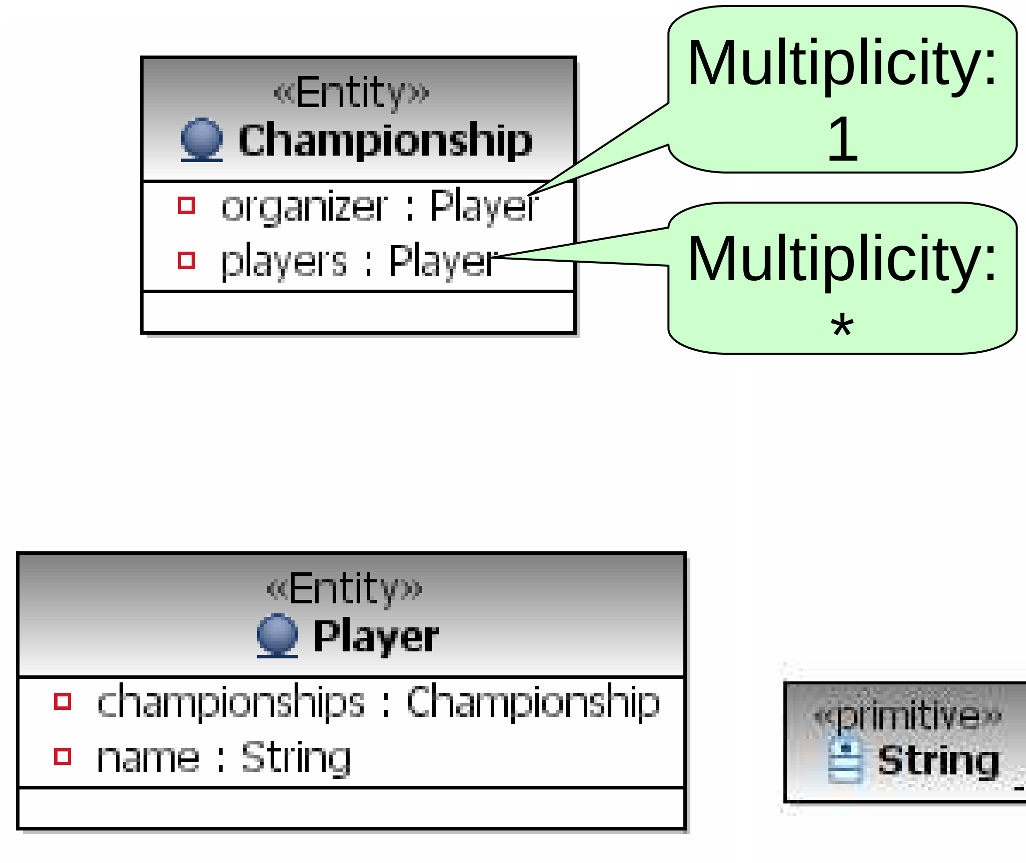
Notation Guide

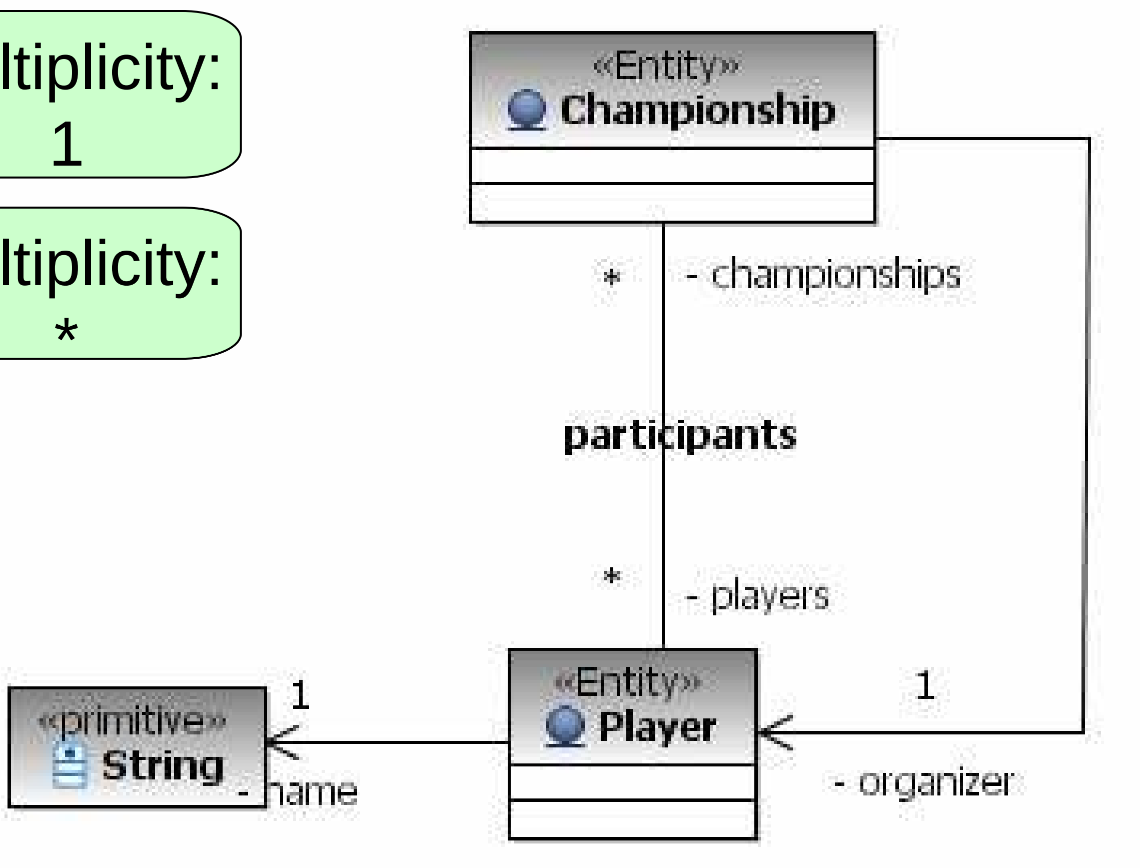


Property = Association + Attribute

Properties as Attributes

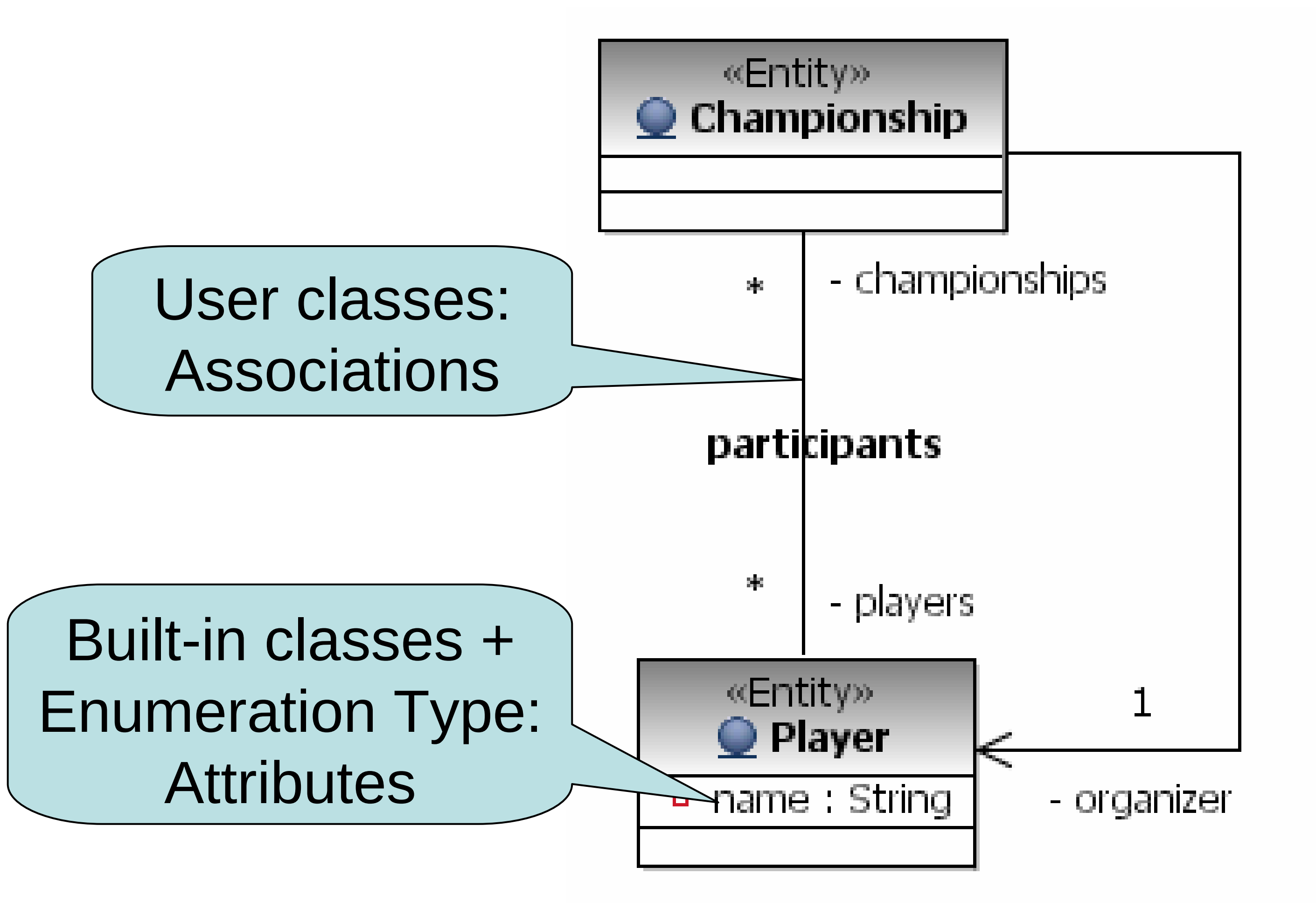
Properties as Associations



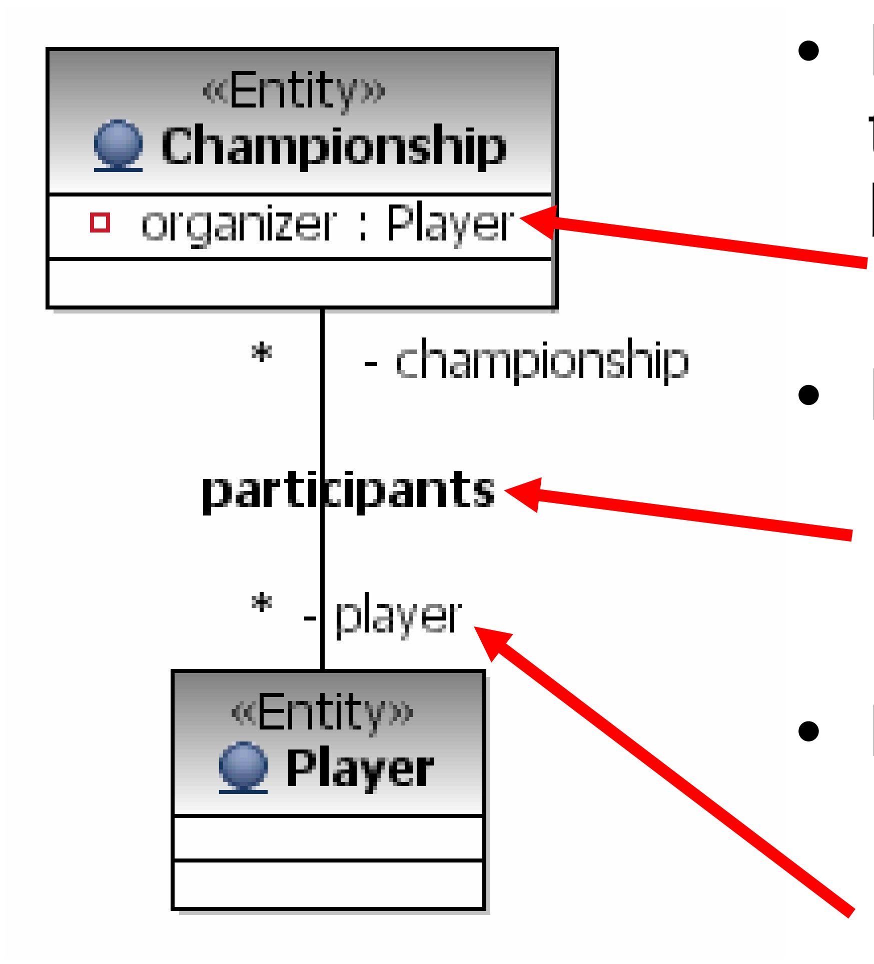


These notations are formally equivalent

Best practice: Properties of Built-in classes vs. User classes

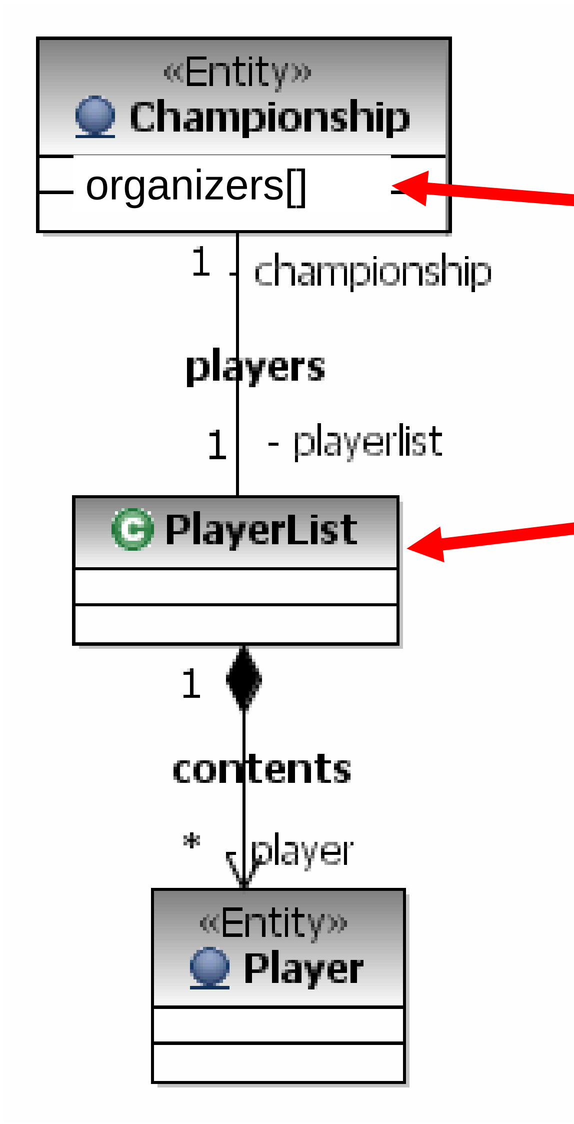


What is Bad Design/Smell here?



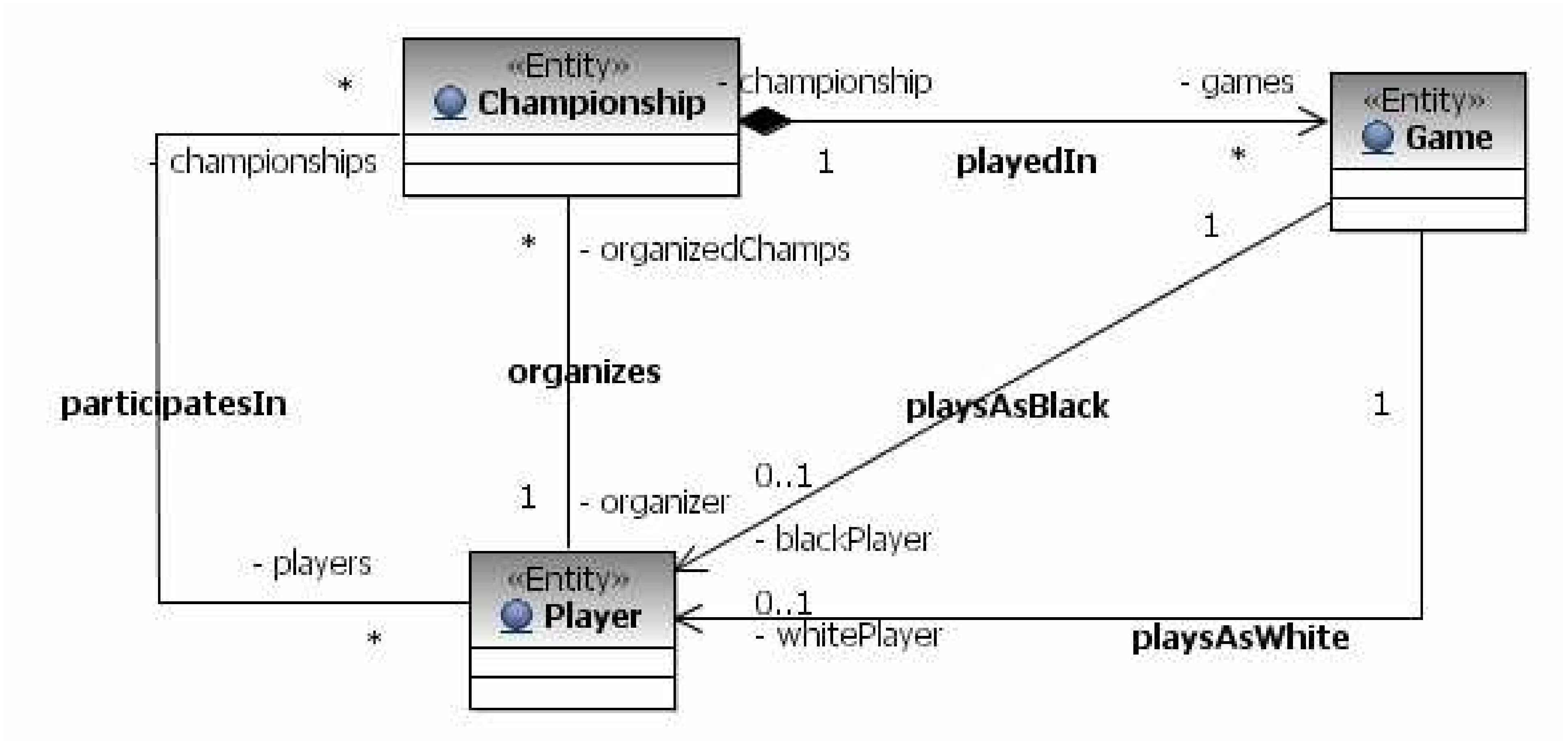
- Properties of a user defined type (class) should rather be denoted explicitly
  - OK, if multiplicity is 1
- Naming of associations:
  - prefer verbs to nouns
  - OK: *participatesIn, participantsOf*
- Naming of roles:
  - 1: singular
  - *: plural
  - OK: *players*, *championships*

What is Bad Design/Smell here?



- Arrays in attributes
  - Solution: an *organizes* association
- Explicit lists
  - Solution: a single *playsIn* association
- NOTE:
  Lists and arrays are programming constructs and not domain elements!

Entity Classes in Championship Management



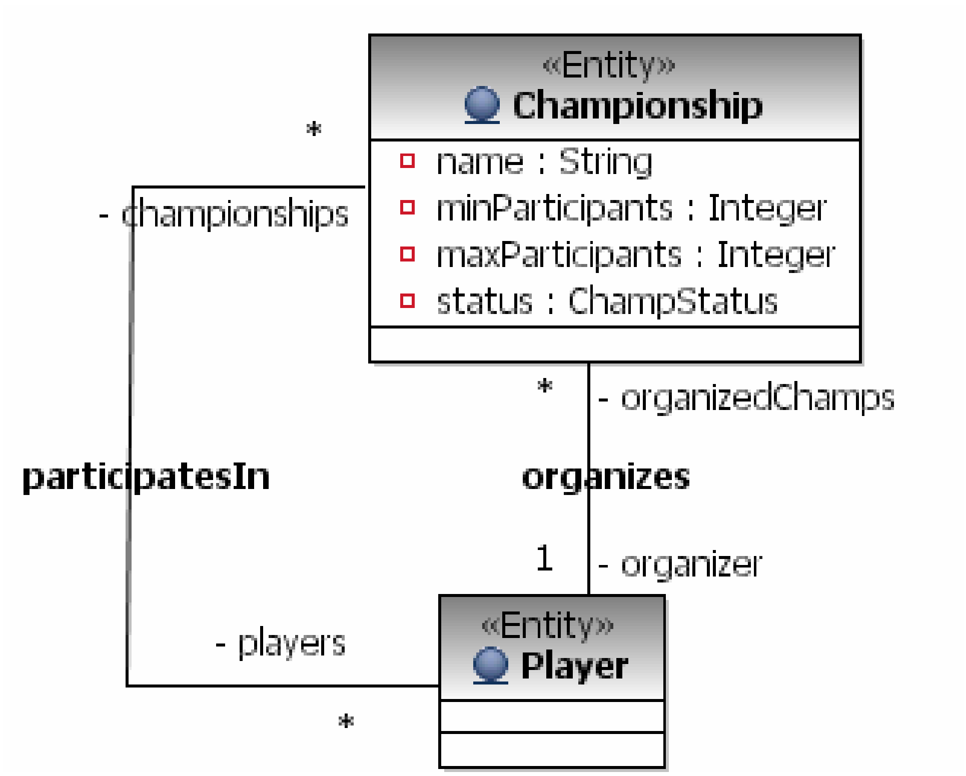
NOTE: Game is not fully defined in this diagram

Mapping of UML Classes to Java

UML	Java
Class	Class
Attribute	Attribute (Field, Prop)
0..1 Association	Attribute (Field, Prop)
0..* Association	Collection<<Class>>
Aggregation	Attribute
Operation	Method
Constraints	Assertions

Implemention in (Pseudo) Java

```
class Championship {
    private String name;
    private Player organizer;
    private Collection players;
}
```



How to set normal attributes?

```
this.setName(newName);
```

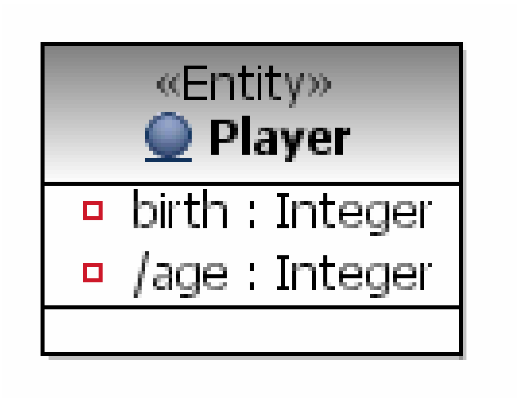
How to set collections?

```
this.getPlayers().add(player);
player.getChampionships().add(this);
```

How to automate?

See a lecture on EMF and code generation

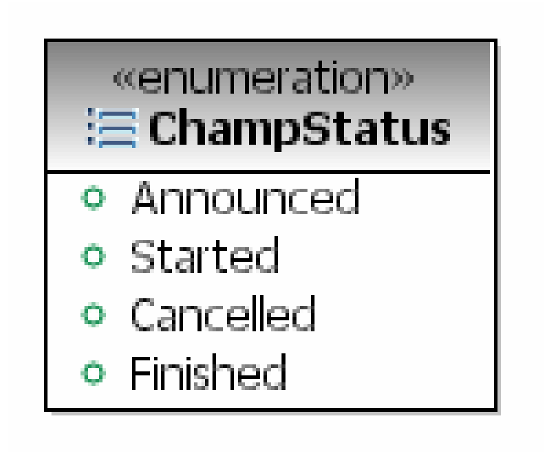
Derived Properties



- A derived property can be calculated from others
- Consequence:
  it need not be persisted
- Example:
  age = currYear - birth

Enumerations

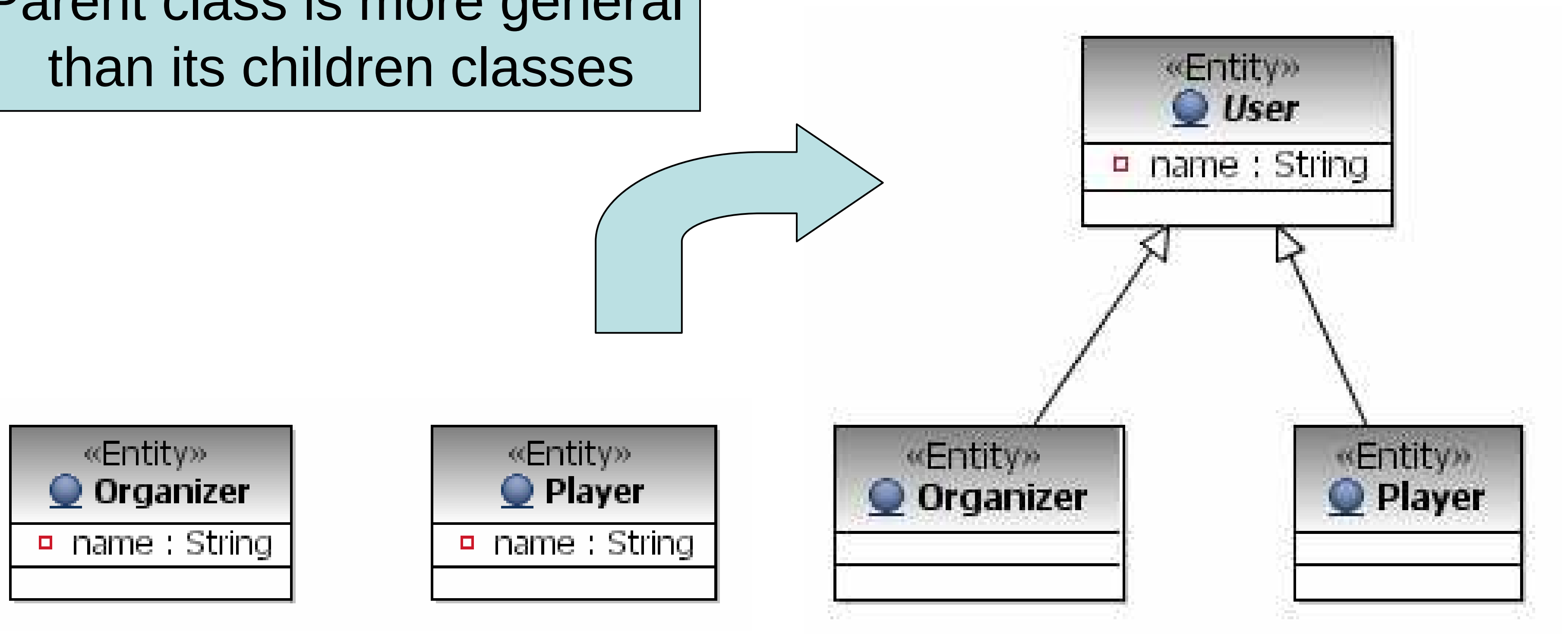
- Enumeration:
  - a fixed set of symbolic values
  - represented as a class with values as attributes
- Usage:
  - Frequently define possible states
  - Use enumerations instead of hard-wired String literals whenever possible



Generalization (Inheritance)

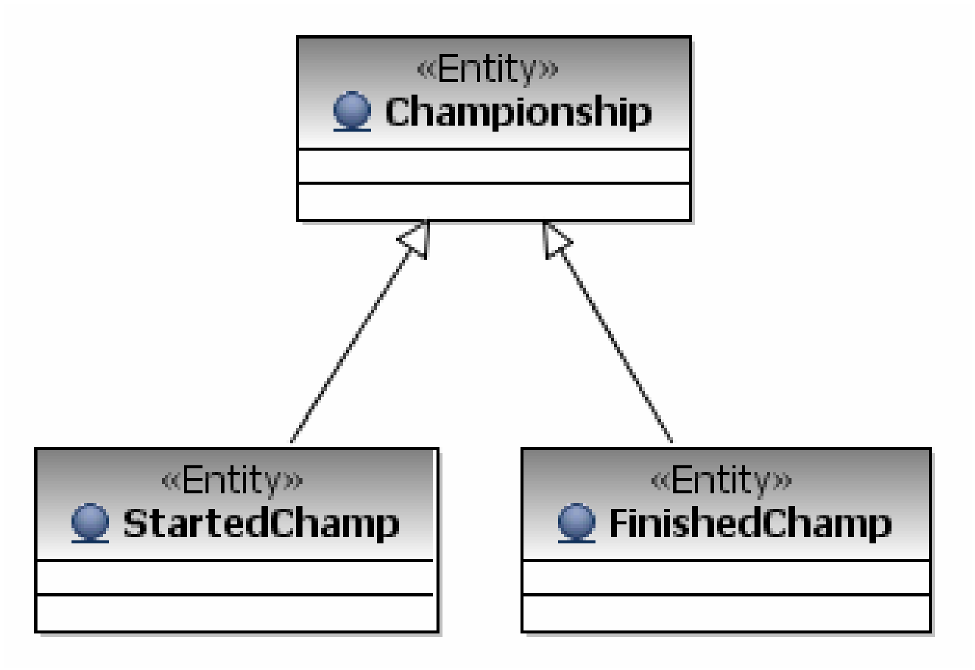
Generalization

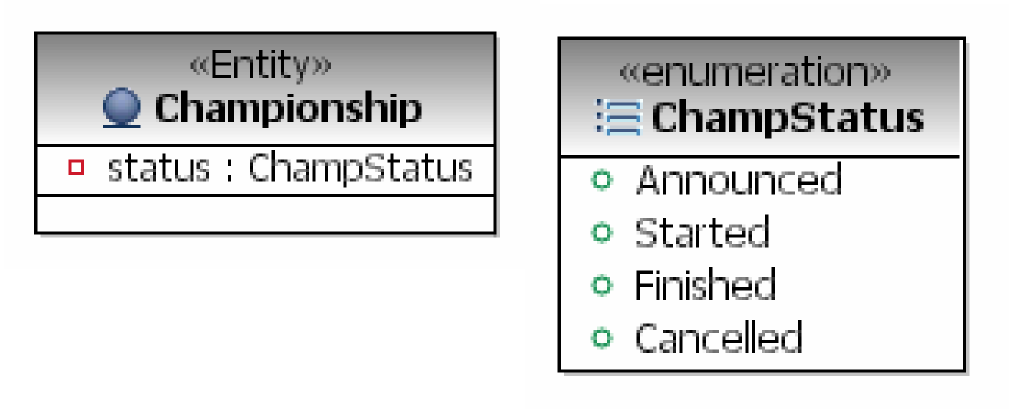
Parent class is more general than its children classes



Aim: Lift up common attributes and methods to the superclass

When to avoid generalization?





- What happens if a started championship is finished?
- Problem: Retyping of an object is required
- NOTE:
  Use status attribute with enumeration values to store the state of an object that can change

Classification vs. Generalization

1. Fido is a Poodle
2. A Poodle is a Dog
3. Dogs are Animals
4. A Poodle is a Breed
5. A Dog is a Species

- ✓ 1+2 = Fido is a Dog
- ✓ 1+2+3 = Fido is an Animal
- ! 1+4 = Fido is a Breed
- ! 2+5 = A Poodle is a Species

Classification vs. Generalization

1. Fido is a Poodle
2. A Poodle is a Dog
3. Dogs are Animals
4. A Poodle is a Breed
5. A Dog is a Species

✓ 1+2 = Fido is a Dog

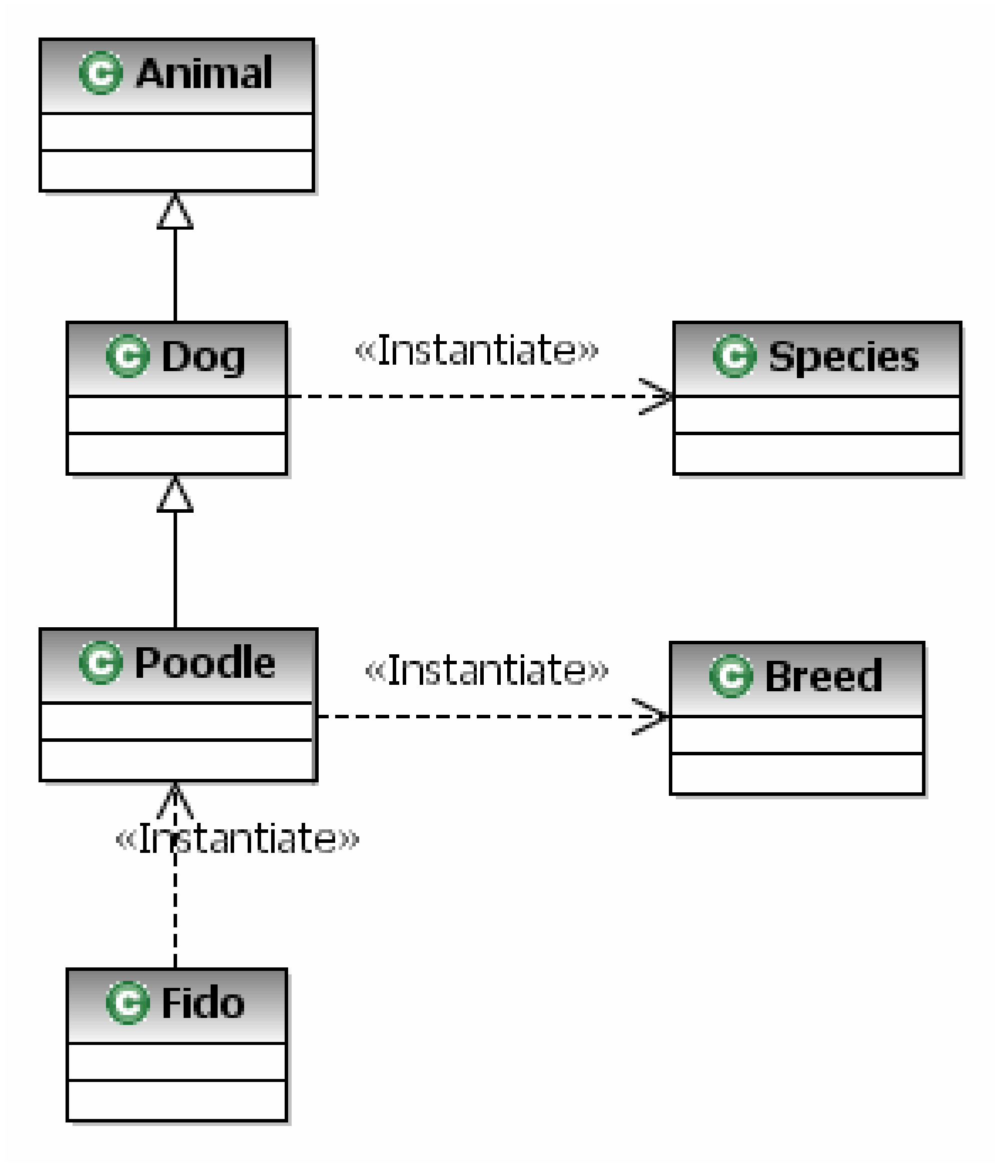
✓ 1+2+3 = Fido is an Animal

! 1+4 = Fido is a Breed

! 2+5 = A Poodle is a Species

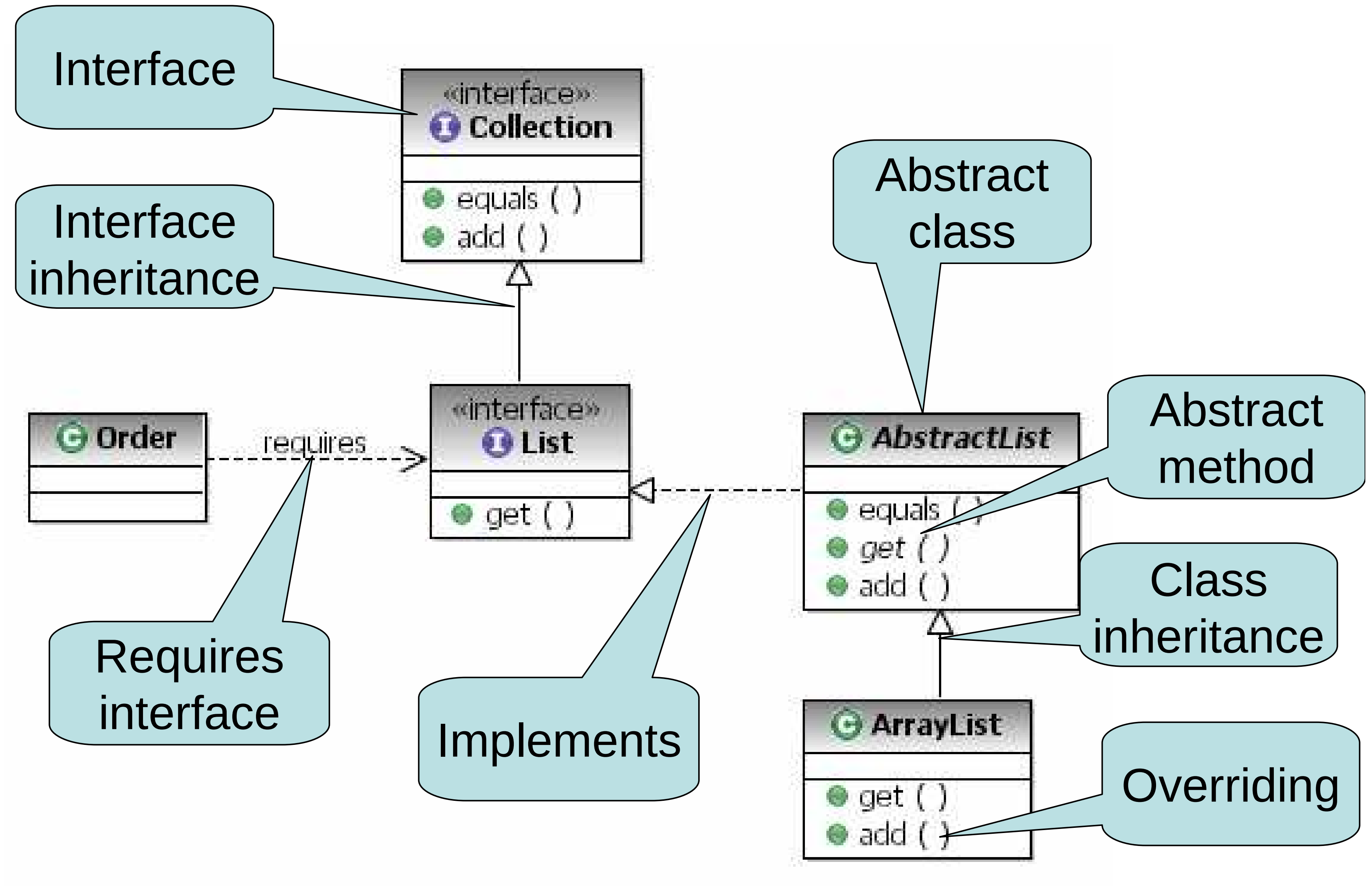
- Generalization (SupertypeOf) is transitive
- Classification (InstanceOf) is NOT transitive

Classification vs. Generalization



Interfaces vs. Abstract Classes

Interfaces vs. Abstract Classes

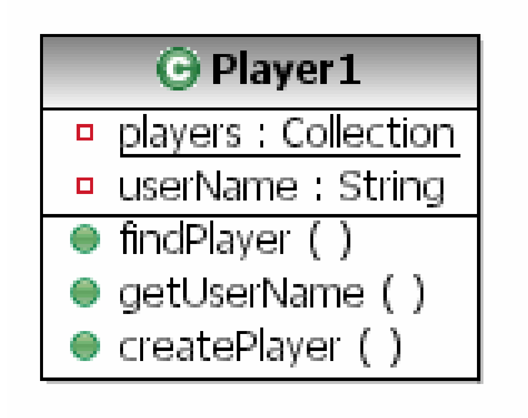


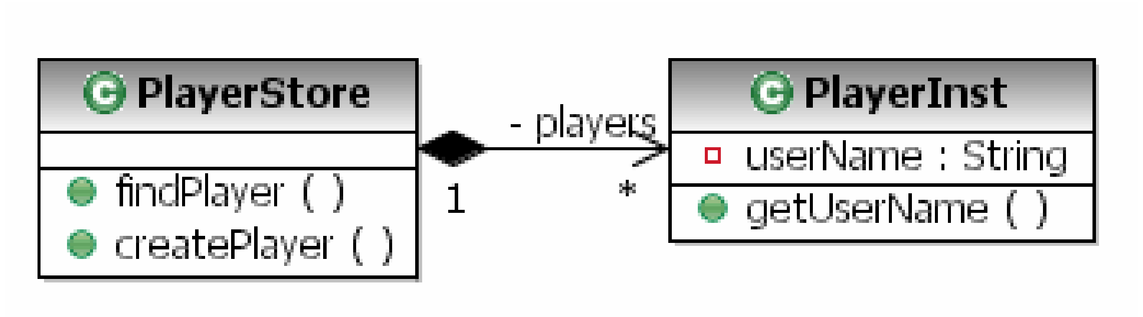
Class-level (Static) Attributes

Example: How to Find a Player

- Use a class-level (static) attribute to store all instances
- Acceptable in pure Java
- NOT in Web apps

- Use a distinct (singleton) container
  - create
  - find
  - delete
- Content
  - Get/Set



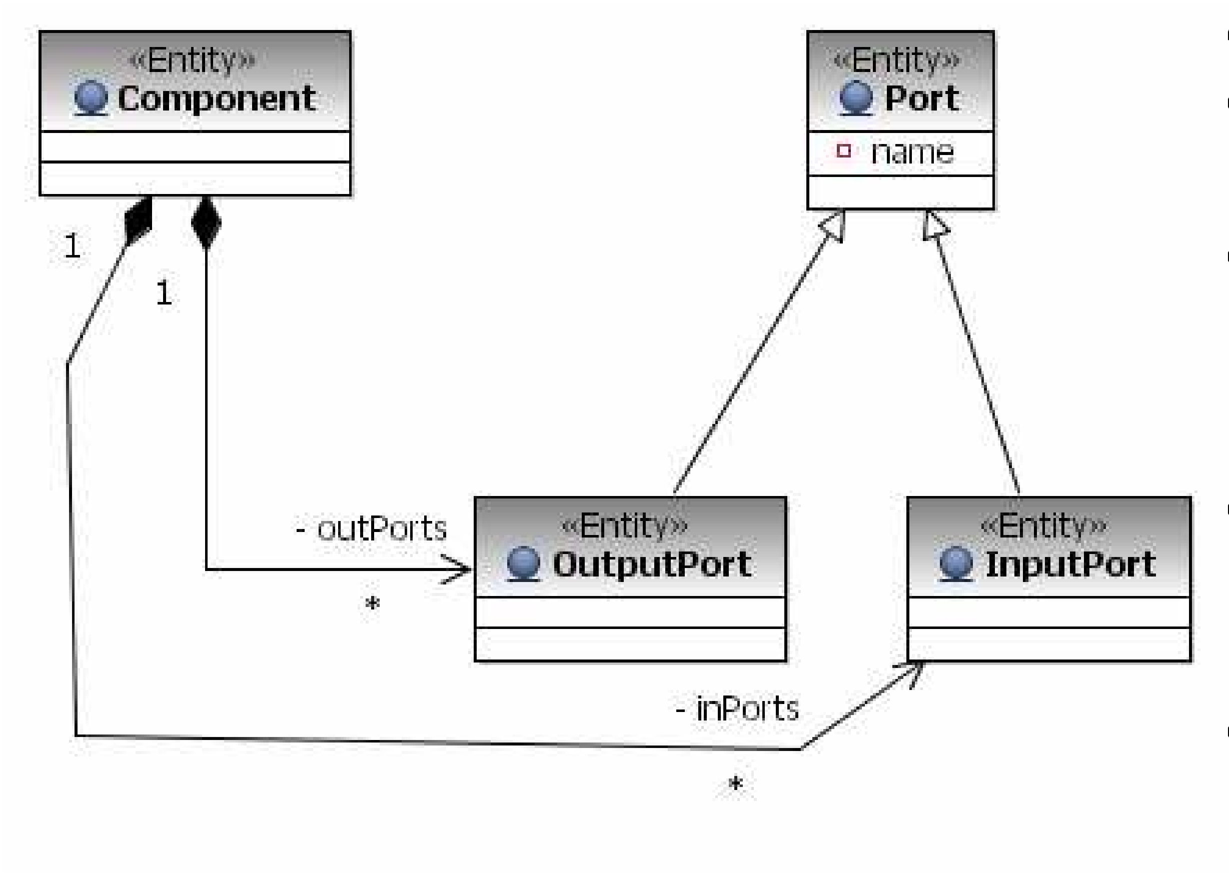


How to Express Restrictions?

A simple modeling problem

- A component aggregates ports with the following restrictions
- Disjointness: a port can be either
  - input ports or
  - output ports
  - but not both
- Completeness:
  All ports are categorized into these two groups
- We should be able to collect input and output ports separately from a component

Restrictions with Generalization



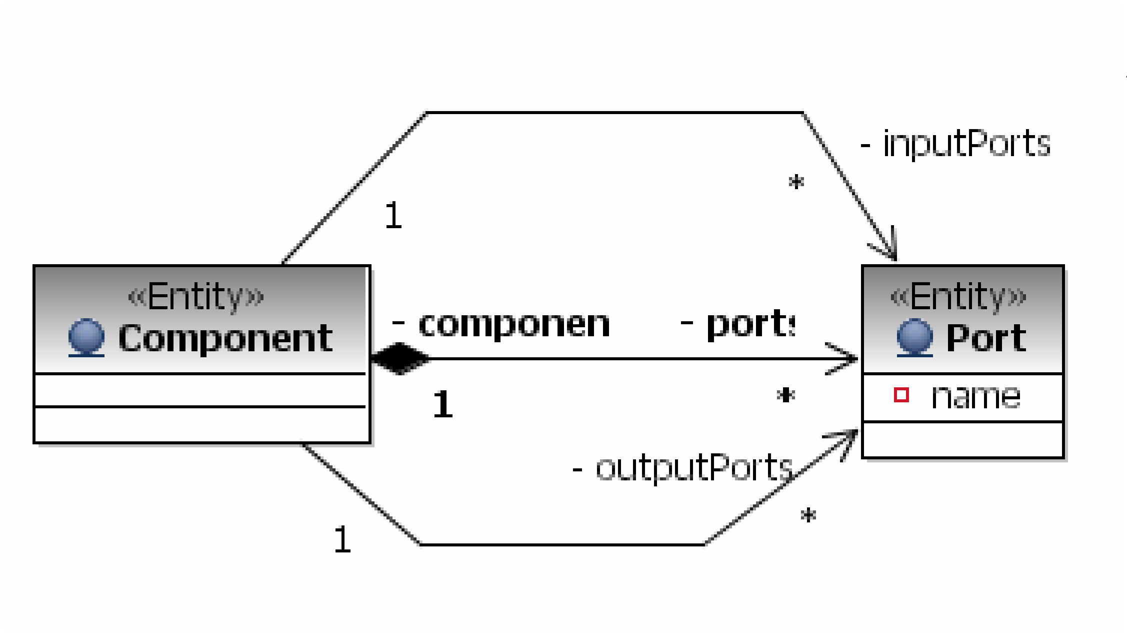
Advantages:

- Input and output ports are disjoint
- Type checking

Disadvantages:

- Type of a port cannot be changed after creation
- Operations common for input and output ports?

Restrictions with (OCL) Constraints



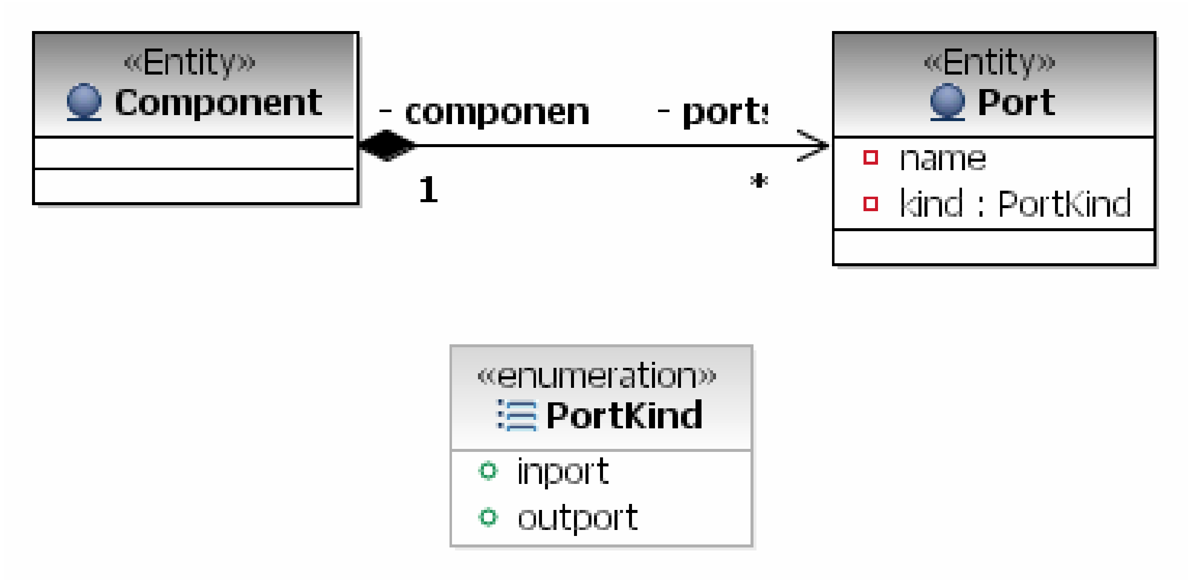
Advantages

- the type of a port can be changed dynamically

Disadvantages:

- constraints are needed to express
  - Disjointness of input and output ports
  - Completeness of input and output ports
- lack of type checking

Restrictions with Enumeration + Attribute



Advantages

- Disjoint
- Complete
- Dynamic changes

Disadvantages

- Access time of in/out ports is increased
- Lack of type checking

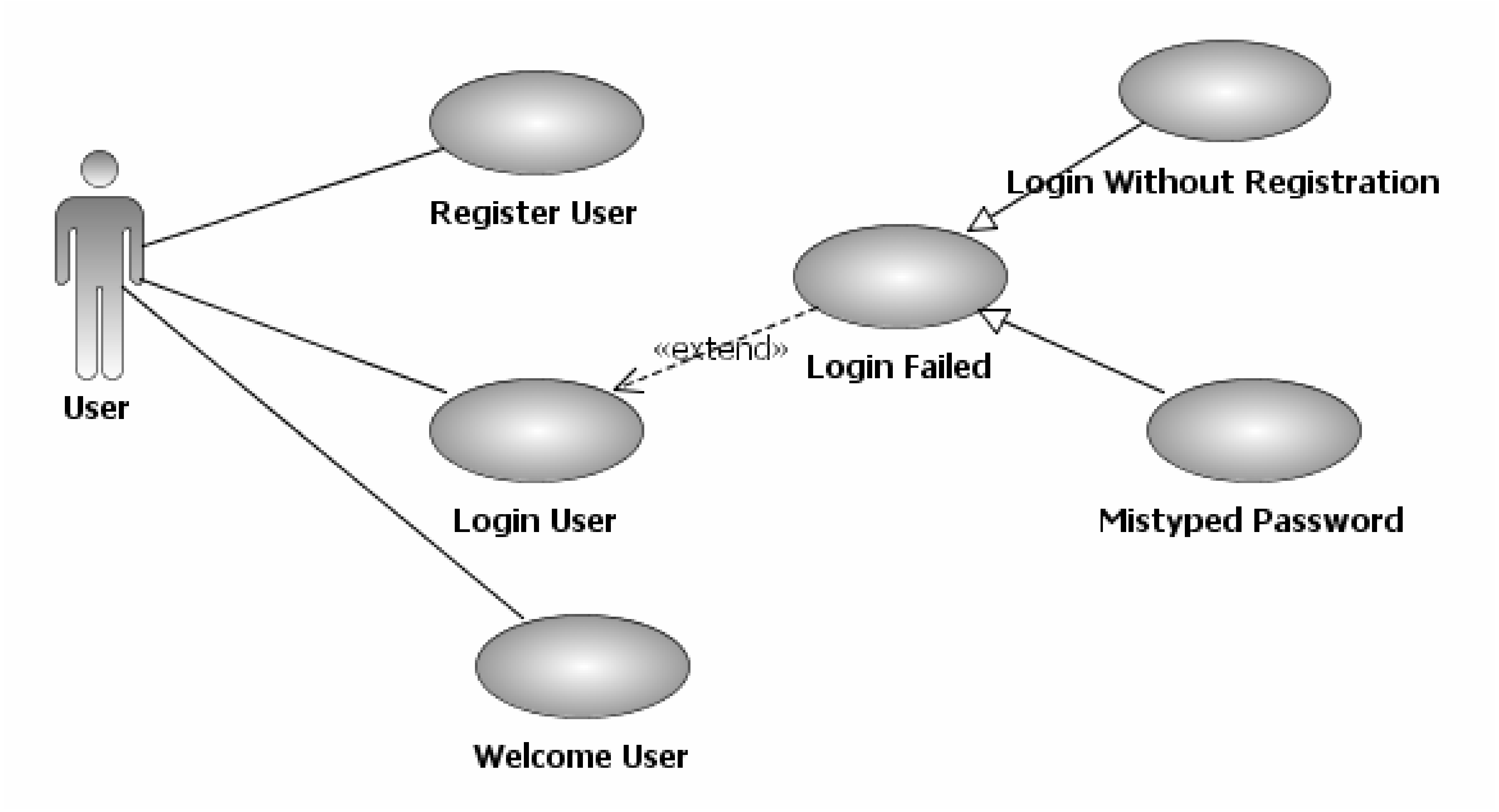
Next Lecture: Interactions

- How to capture flows of interaction (scenarios)?

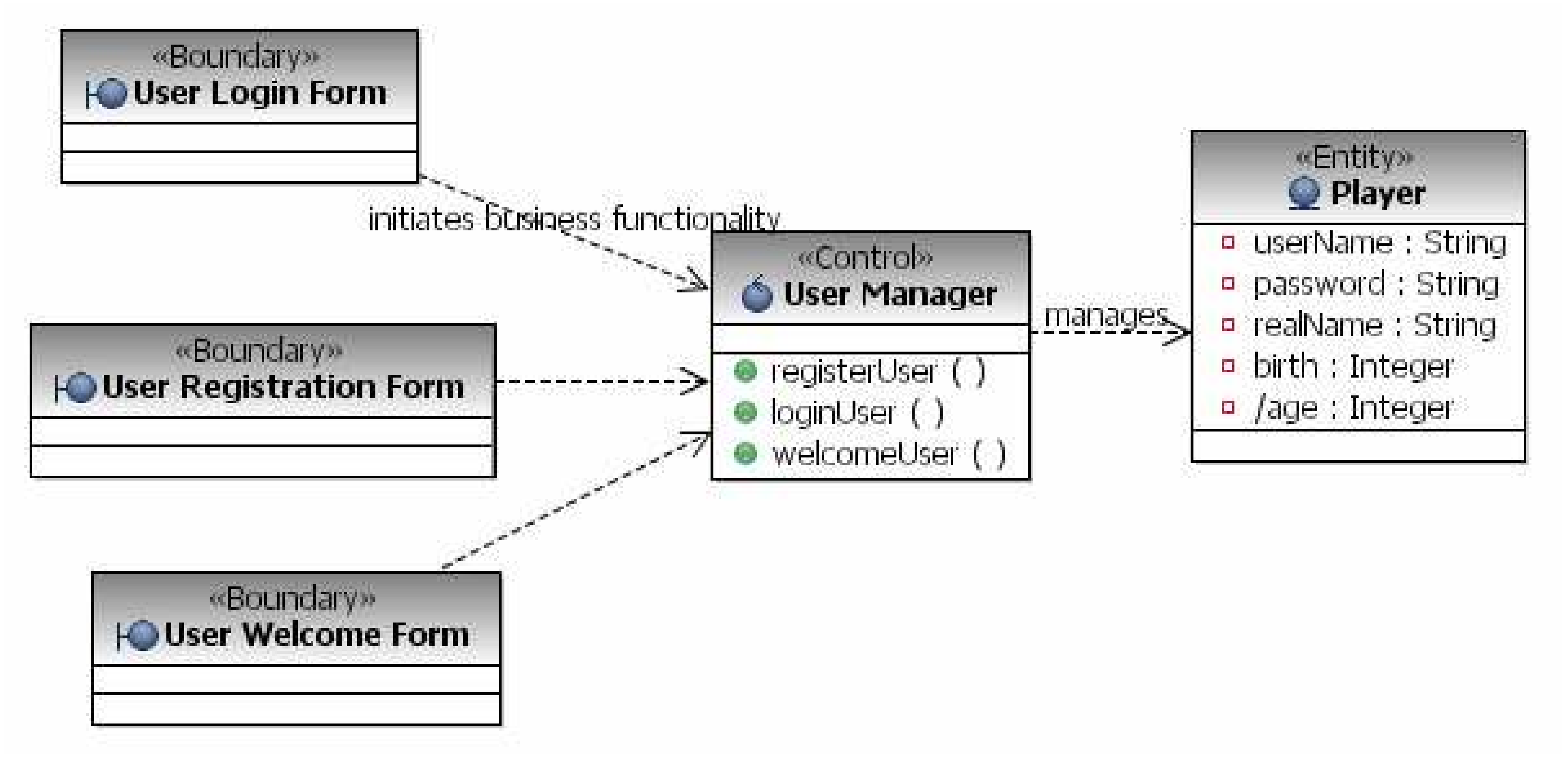
- How do analysis classes interact?

Milestone: Analysis Classes for Championship Manager

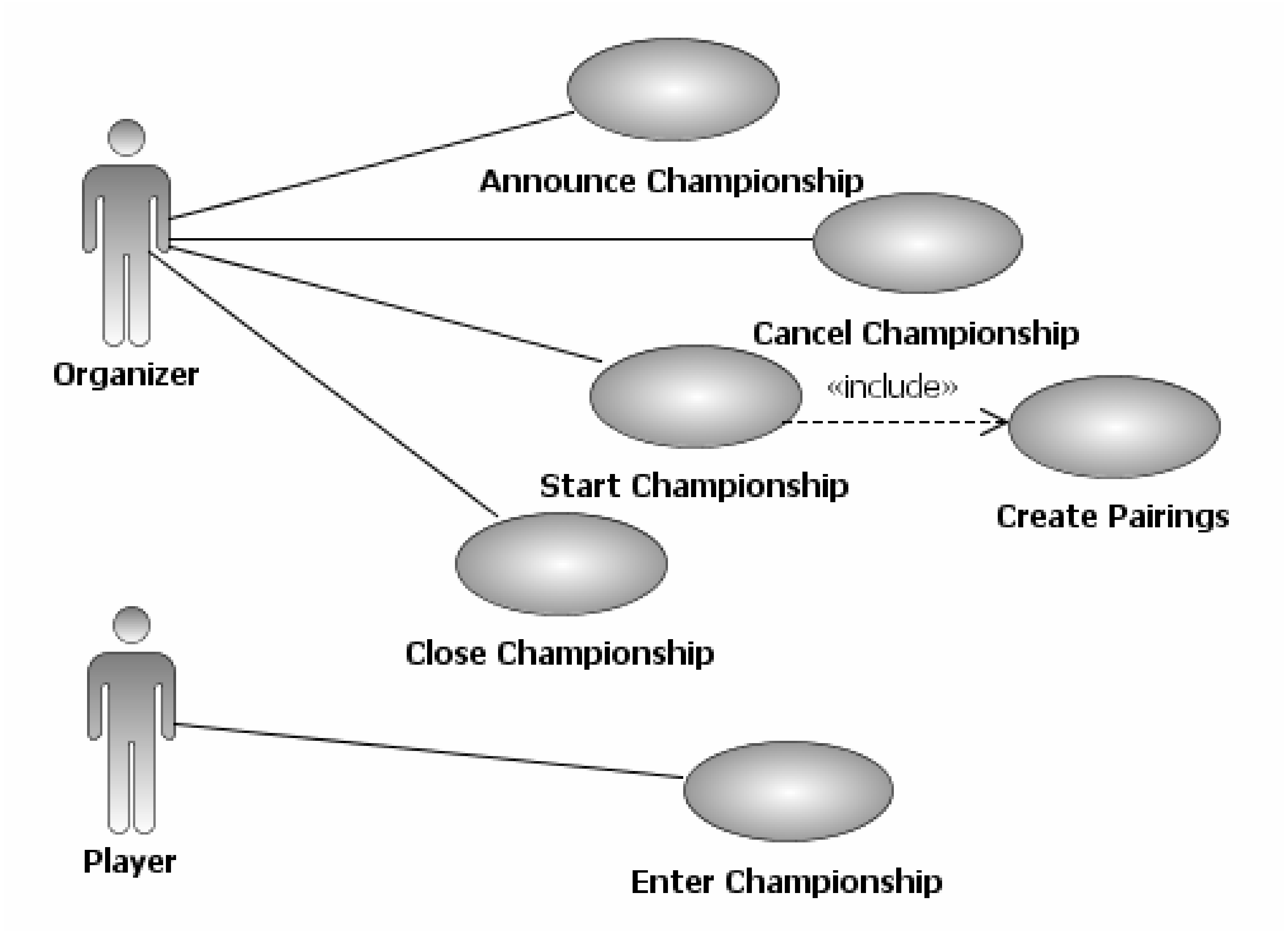
User Management Use Cases



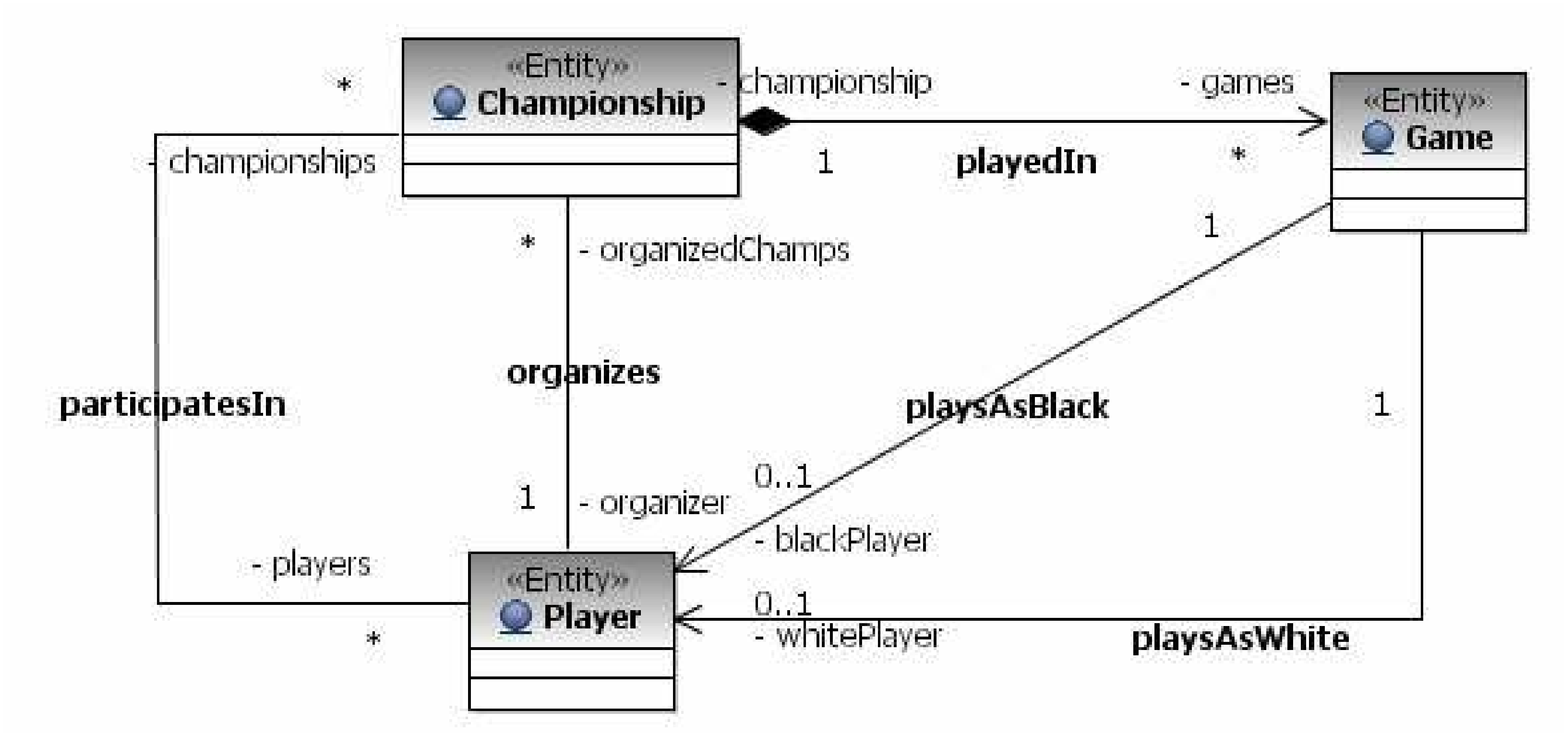
User Management Analysis Classes



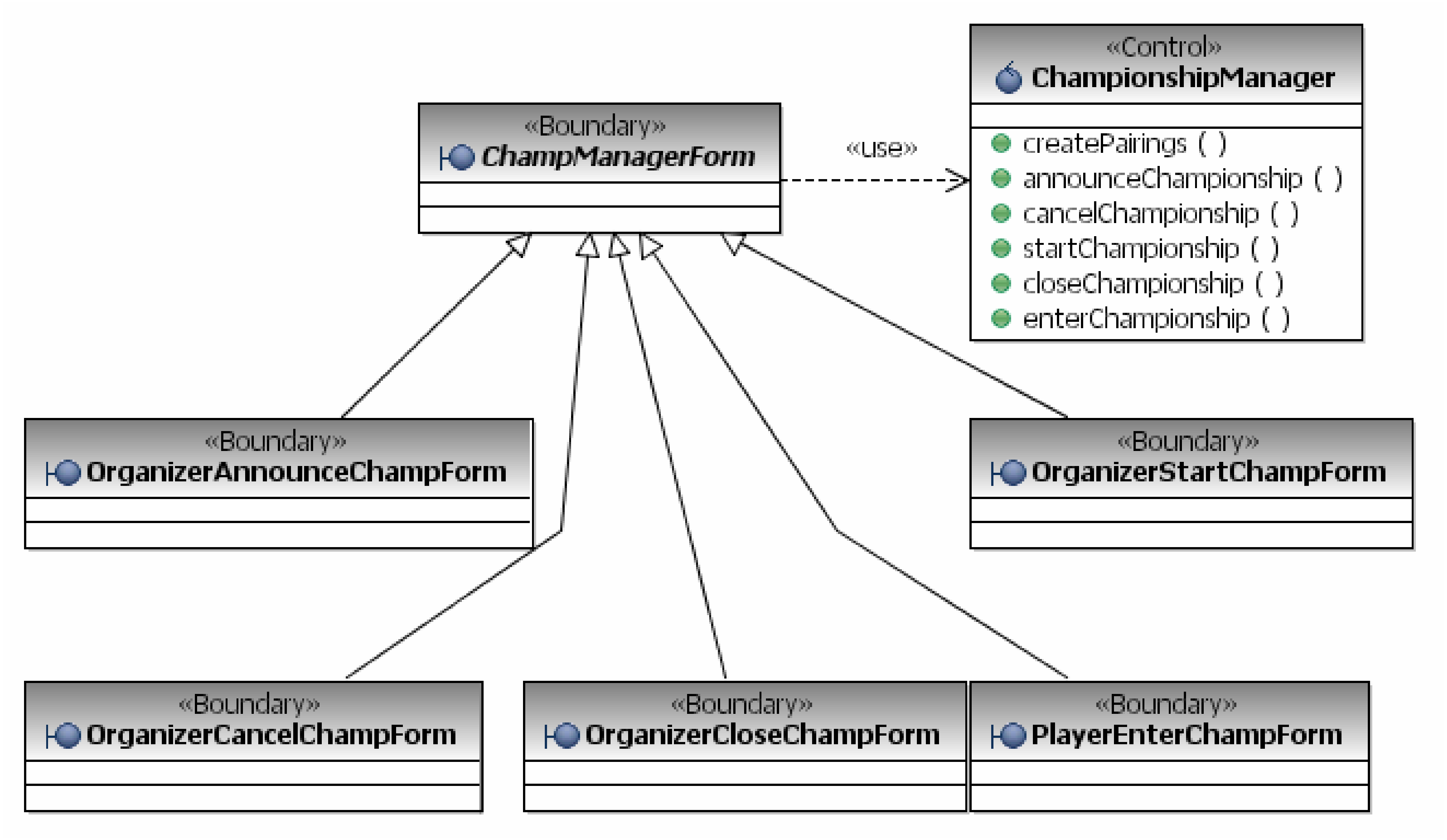
Championship Management



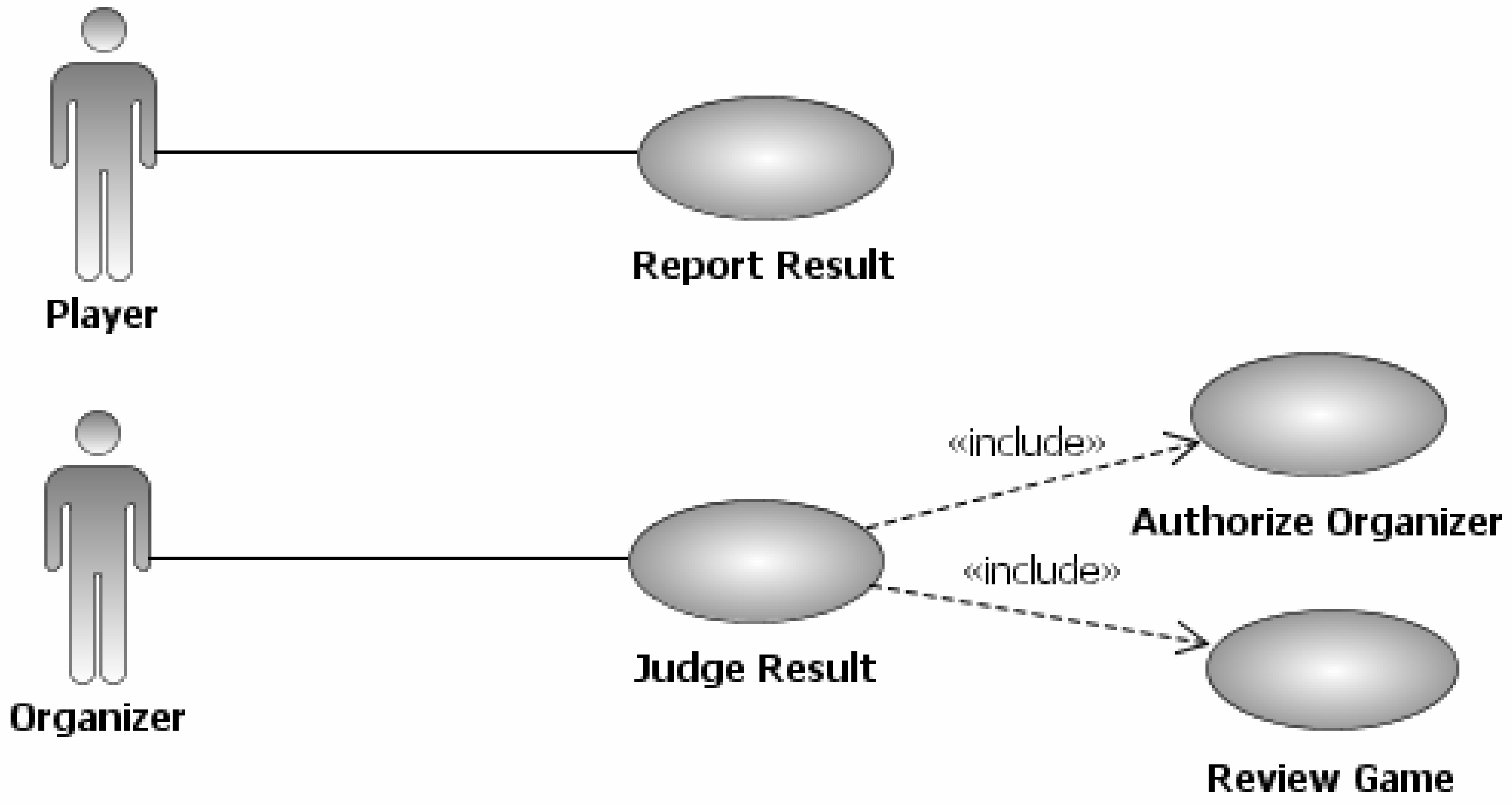
Entity Classes in Championship Management



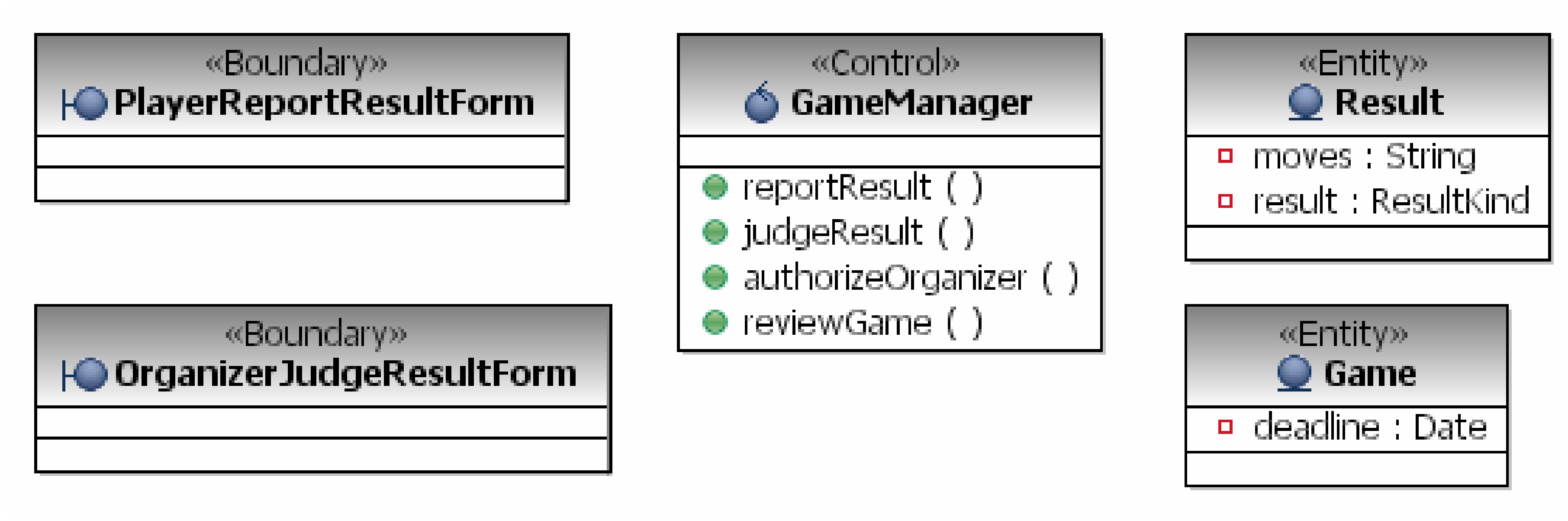
Championship Manager: Control and Boundary Classes



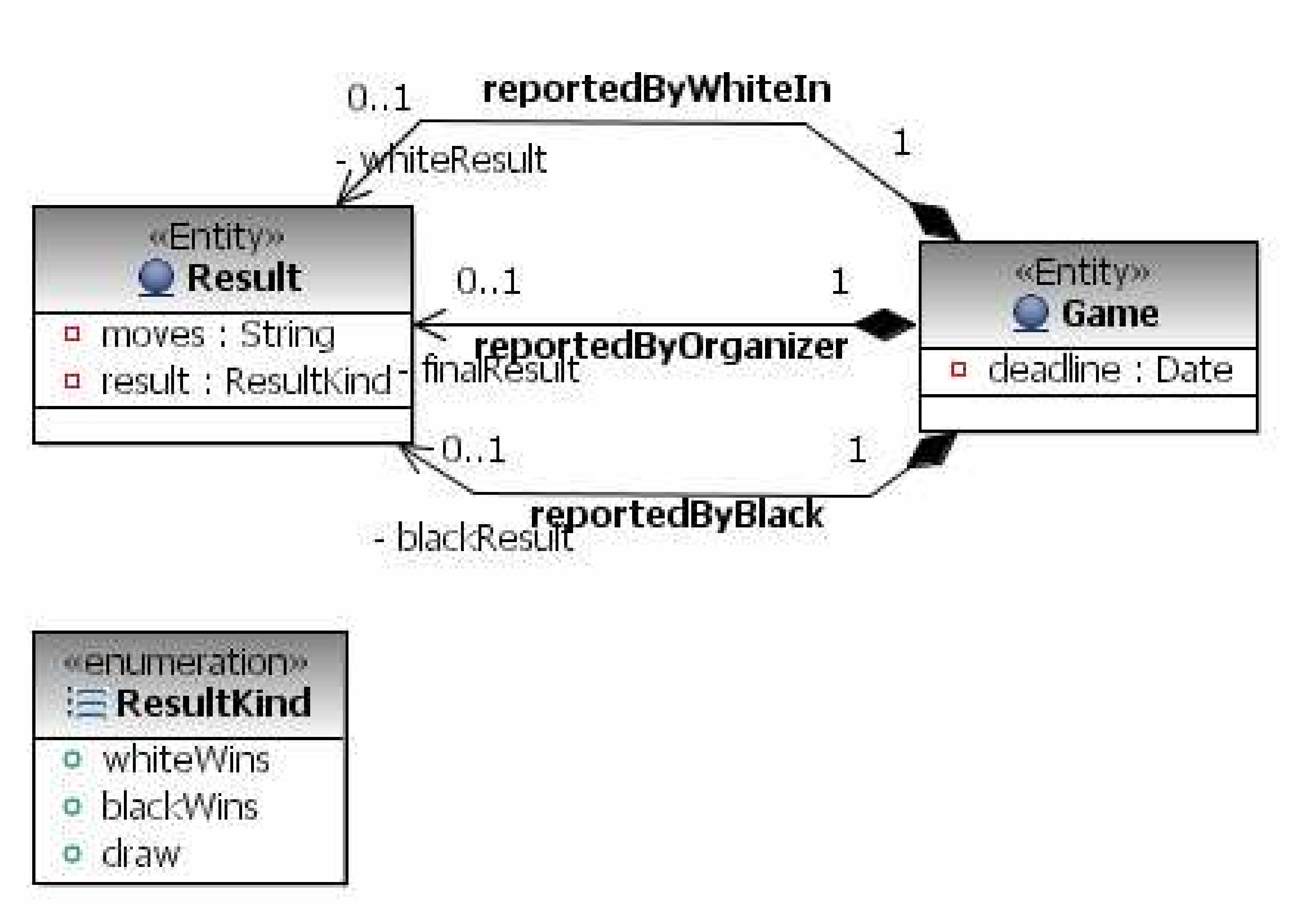
Game Management Use Cases



Game Management Analysis Classes



Game Management Entity Classes



Példányosítás vs. Öröklés

- Fifi egy uszkár
- Az uszkár egy kutya
- A kutya állat
- Az uszkár egy fajta
- A kutya egy faj