Chapter 4 High-level Database Models

- Entity/Relationship Models (E/R diagram)
- Unified Modeling Language (UML)
- Object Definition Language (ODL)
- Hot to Transfer them to a relational model

Other High-Level Design Languages

Object Definition Language (ODL)

Unified Modeling Language (UML)

Object-Oriented DBMS’s

- Standards group: ODMG = Object Data Management Group.
- ODL = Object Description Language, like CREATE TABLE part of SQL.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.

Framework – (1)

- ODMG imagines OO-DBMS vendors implementing an OO language like C++ with extensions (OQL) that allow the programmer to transfer data between the database and “host language” seamlessly.

Framework – (2)

- ODL is used to define *persistent* classes, whose objects are stored permanently in the database.
  - ODL classes look like Entity sets with binary relationships, plus methods.
  - ODL class definitions are part of the extended, OO host language.

ODL Overview

- A class declaration includes:
  1. A name for the class.
  2. Optional key declaration(s).
  3. Element declarations. An *element* is either an attribute, a relationship, or a method.

Class Definitions

class <name> {

<list of element declarations, separated

by semicolons>

}

Attribute and Relationship Declarations

- Attributes are (usually) elements with a type that does not involve classes.

  ```
  attribute <type> <name>;
  ```

- Relationships connect an object to one or more other objects of one class.

  ```
  relationship <type> <name>
      inverse <relationship>;
  ```

Inverse Relationships

- Suppose class *C* has a relationship *R* to class *D*.
- Then class *D* must have some relationship *S* to class *C*.
- *R* and *S* must be true inverses.
  - If object *d* is related to object *c* by *R*, then *c* must be related to *d* by *S*.

Example: Attributes and Relationships

```
class Bar {
    attribute string name;
    attribute string addr;
    relationship Set<Beer> serves inverse Beer::servedAt;
}
class Beer {
    attribute string name;
    attribute string manf;
    relationship Set<Bar> servedAt inverse Bar::serves;
}
```

The type of relationship serves is a set of Beer objects.

The :: operator connects a name on the right to the context containing that name, on the left.

Types of Relationships

- The type of a relationship is either
  1. A class, like Bar. If so, an object with this relationship can be connected to only one Bar object.
  2. Set<Bar>: the object is connected to a set of Bar objects.
  3. Bag<Bar>, List<Bar>, Array<Bar>: the object is connected to a bag, list, or array of Bar objects.

Multiplicity of Relationships

- All ODL relationships are binary.
- Many-many relationships have Set<…> for the type of the relationship and its inverse.
- Many-one relationships have Set<…> in the relationship of the "one" and just the class for the relationship of the "many."
- One-one relationships have classes as the type in both directions.

Example: Multiplicity

```
class Drinker { …
    relationship Set<Beer> likes inverse Beer::fans;
    relationship Beer favBeer inverse Beer::superfans;
}
class Beer { …
    relationship Set<Drinker> fans inverse Drinker::likes;
    relationship Set<Drinker> superfans inverse
    Drinker::favBeer;
}
```

Many-many uses Set<…> in both directions.

Many-one uses Set<…> only with the "one."

Another Multiplicity Example

husband and wife are one-one and inverses of each other.

```
class Drinker {
    attribute … ;
    relationship Drinker husband inverse wife;
    relationship Drinker wife inverse husband;
    relationship Set<Drinker> buddies
        inverse buddies;
}
```

buddies is many-many and its own inverse. Note no :: needed if the inverse is in the same class.

Coping With Multiway Relationships

- ODL does not support 3-way or higher relationships.
- We may simulate multiway relationships by a "connecting" class, whose objects represent tuples of objects we would like to connect by the multiway relationship.

Connecting Classes

- Suppose we want to connect classes *X*, *Y*, and *Z* by a relationship *R*.
- Devise a class *C*, whose objects represent a triple of objects (*x*, *y*, *z*) from classes *X*, *Y*, and *Z*, respectively.
- We need three many-one relationships from (*x*, *y*, *z*) to each of *x*, *y*, and *z*.

Example: Connecting Class

- Suppose we have Bar and Beer classes, and we want to represent the price at which each Bar sells each beer.
  - A many-many relationship between Bar and Beer cannot have a price attribute as it did in the E/R model.
- One solution: create class Price and a connecting class BBP to represent a related bar, beer, and price.

Example -- Continued

- Since Price objects are just numbers, a better solution is to:
  1. Give BBP objects an attribute price.
  2. Use two many-one relationships between a BBP object and the Bar and Beer objects it represents.

Example -- Concluded

- Here is the definition of BBP:

```
class BBP {
    attribute price:real;
    relationship Bar theBar inverse Bar::toBBP;
    relationship Beer theBeer inverse
    Beer::toBBP;
}
```

- Bar and Beer must be modified to include relationships, both called toBBP, and both of type Set<BBP>.

Structs and Enums

- Attributes can have a structure (as in C) or be an enumeration.
- Declare with

attribute [Struct or Enum] <name of struct or enum> { <details> } <name of attribute>;

- Details are field names and types for a Struct, a list of constants for an Enum.

Example: Struct and Enum

Names for the structure and enumeration

```
class Bar {
    attribute string name;
    attribute Struct Addr
    {string street, string city, int zip} address;
    attribute Enum Lic
        { FULL, BEER, NONE } license;
    relationship …
}
```

names of the attributes

Method Declarations

- A class definition may include declarations of methods for the class.
- Information consists of:
  1. Return type, if any.
  2. Method name.
  3. Argument modes and types (no names).
     - Modes are in, out, and inout.
  4. Any exceptions the method may raise.

Example: Methods

```
real gpa(in string)raises(noGrades);
```

1. The method gpa returns a real number (presumably a student's GPA).
2. gpa takes one argument, a string (presumably the name of the student) and does not modify its argument.
3. gpa may raise the exception noGrades.

The ODL Type System

- Basic types: int, real/float, string, enumerated types, and classes.
- Type constructors:
  - Struct for structures.
  - *Collection types* : Set, Bag, List, Array, and Dictionary (= mapping from a domain type to a range type).
- Relationship types can only be a class or a single collection type applied to a class.

ODL Subclasses

- Usual object-oriented subclasses.
- Indicate superclass with a colon and its name.
- Subclass lists only the properties unique to it.
  - Also inherits its superclass' properties.

Example: Subclasses

- Ales are a subclass of beers:

```
class Ale:Beer {
   attribute string color;
}
```

ODL Keys

- You can declare any number of keys for a class.
- After the class name, add:

  (key <list of keys>)
- A key consisting of more than one attribute needs additional parentheses around those attributes.

Example: Keys

```
class Beer (key name) { …
```

- name is the key for beers.

```
class Course (key
   (dept,number),(room, hours)){
```

- dept and number form one key; so do room and hours.

Translating ODL to Relations

- Classes without relationships: like entity set, but several new problems arise.
- Classes with relationships:

a) Treat the relationship separately, as in E/R.

b) Attach a many-one relationship to the relation for the “many”.

ODL Class Without Relationships

- Problem: ODL allows attribute types built from structures and collection types.
- Solutions:

1. Structure: Make one attribute for each field.
2. Set: make one tuple for each member of the set. More than one set attribute ? Make tuples for all combinations.

- Problem: ODL class may have no key, but we should have one in the relation to represent " OID".

Example

Class Drinkers (key name)

{ attribute string name;

attribute Struct Addr { string street, string city, int zip} address;

attribute Set <string> phone; }

Name	street	city	zip	phone
n_1	s_1	c_1	z_1	p_1
n_1	s_1	c_1	z_1	p_2

Example (cont.)

- Surprise: the key for class (name) is not the key for the relation (name, phone)
- Name in the class determines a unique object, including a set of phones.
- Name in the relation does not determine a unique tuple.
- Since tuples are not identical to objects, there is no inconsistency!
- BCNF violation: separate out name-phone.

ODL Relationships

- Create for each relationship a new relation that connects the keys of the two related classes, one relation for each pair.
- If the relationship is many-one from A to B, put key of B attributes in the relation for class A.

Example

```
Class Drinkers (key name) {
  attribute string name;
  attribute string addr;
  relationship Set<Beers> likes inverse Beers:: fans;
  relationship Beers favorite inverse Beers:: realFans;
  Relationship Drinkers husband inverse wife;
  Relationship Drinkers wife inverse husband;
  Relationship Set<Drinkers> buddies inverse buddies;
 }
```

Drinkers (<u>name</u>, addr, favBeer, marriedwith)

Likes (drinkerName, Beersname)

Buddy (drinker1,drinker2)

UML introduction

- UML is an acronym for Unified Modeling Language.

- The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a software-intensive system.

- Object-Oriented & Visual Modeling

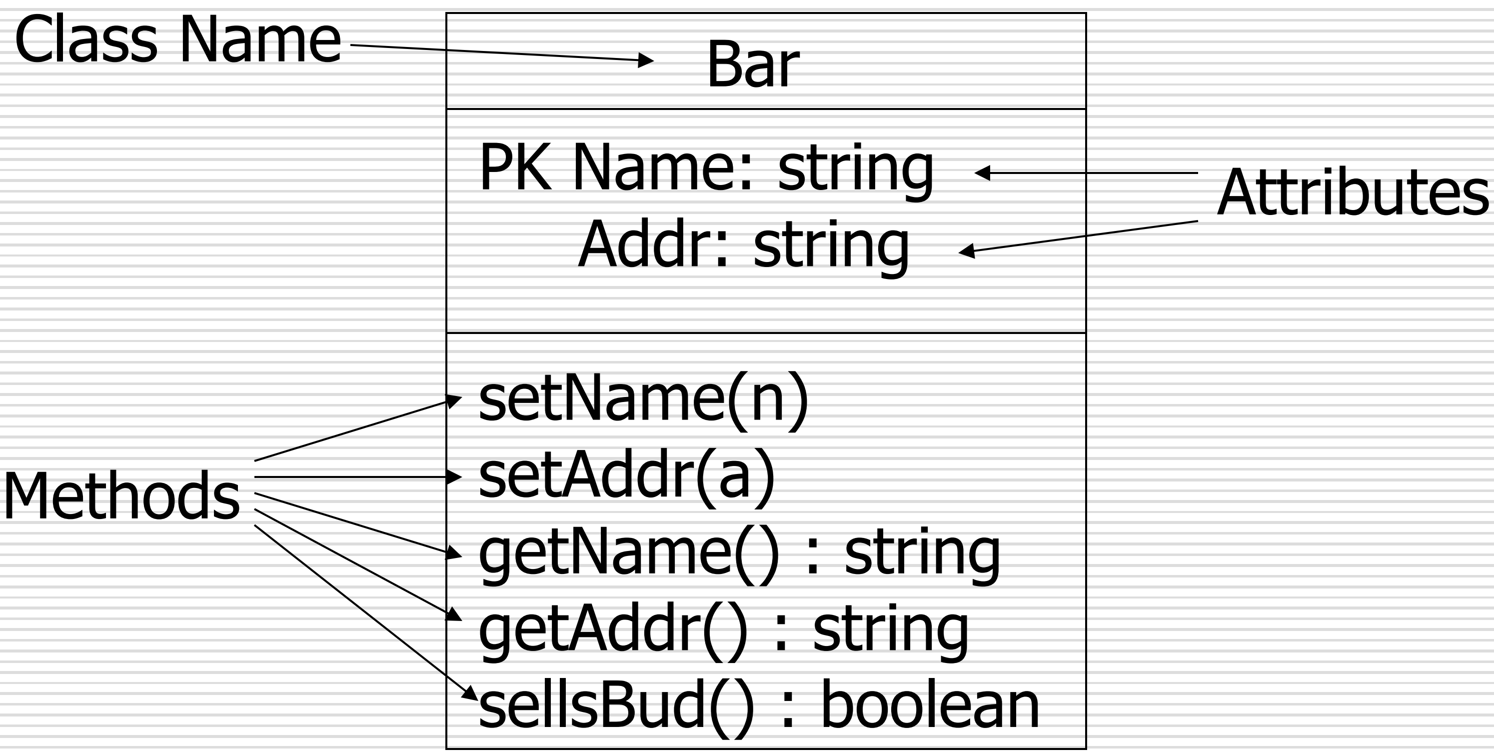
UML

- UML is designed to model software, but has been adapted as a database modeling language.
- Midway between E/R and ODL.
  - No multiway relationships as in E/R.
  - But allows attributes on binary relationships, which ODL doesn't.
  - Has a graphical notation, unlike ODL.

Classes

- Sets of objects, with attributes (*state*) and methods (*behavior*).
- Attributes have types.
- PK indicates an attribute in the primary key (optional) of the object.
- Methods have declarations: arguments (if any) and return type.

Example: Bar Class



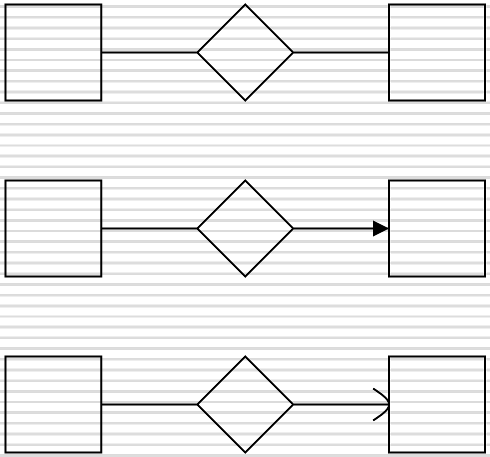
Associations

- Binary relationships between classes.
- Represented by named lines (no diamonds as in E/R).
- Multiplicity at each end.
  - *m* ..*n* means between *m* and *n* of these associate with one on the other end.
  - * = "infinity"; e.g. 1..* means "at least one."

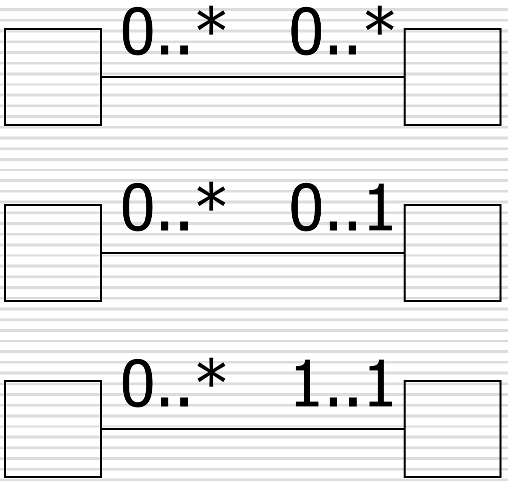
Example: Association

Comparison With E/R Multiplicities

E/R



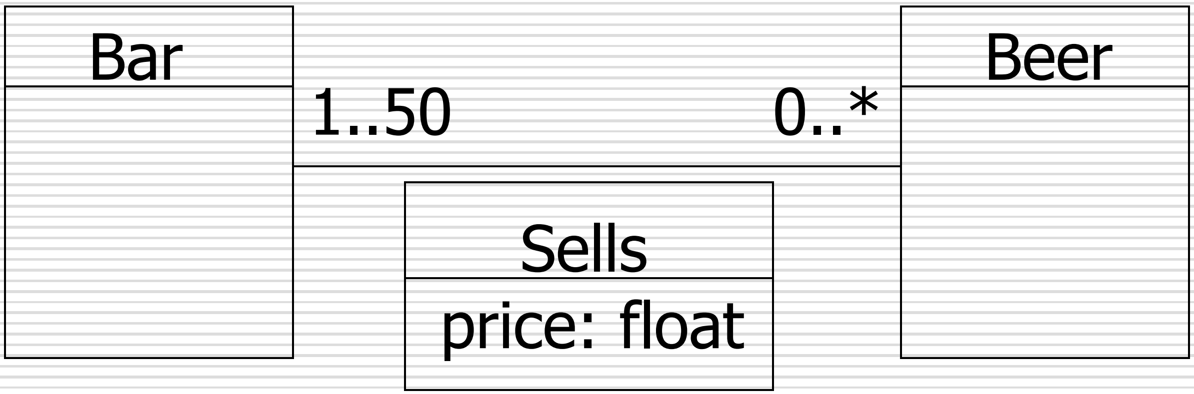
UML



Association Classes

- Attributes on associations are permitted.
  - Called an *association class*.
  - Analogous to attributes on relationships in E/R.

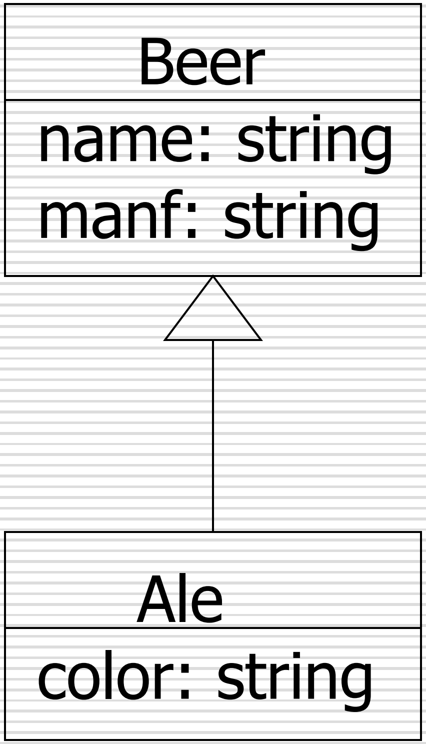
Example: Association Class



Subclasses

- Like E/R, but subclass points to superclass with a line ending in a triangle.
- The subclasses of a class can be:
  - *Complete* (every object is in at least one subclass) or *partial*.
  - *Disjoint* (object in at most one subclass) or *overlapping*.

Example: Subclasses in UML



Subclasses (cont.)

- In a typical object-oriented system, subclasses are disjoint.
- E/R model allows overlapping subclasses.
- E/R model and object-oriented system allow either complete or partial subclasses. There is no requirement that a member of the superclass be in any of subclass.

Aggregations

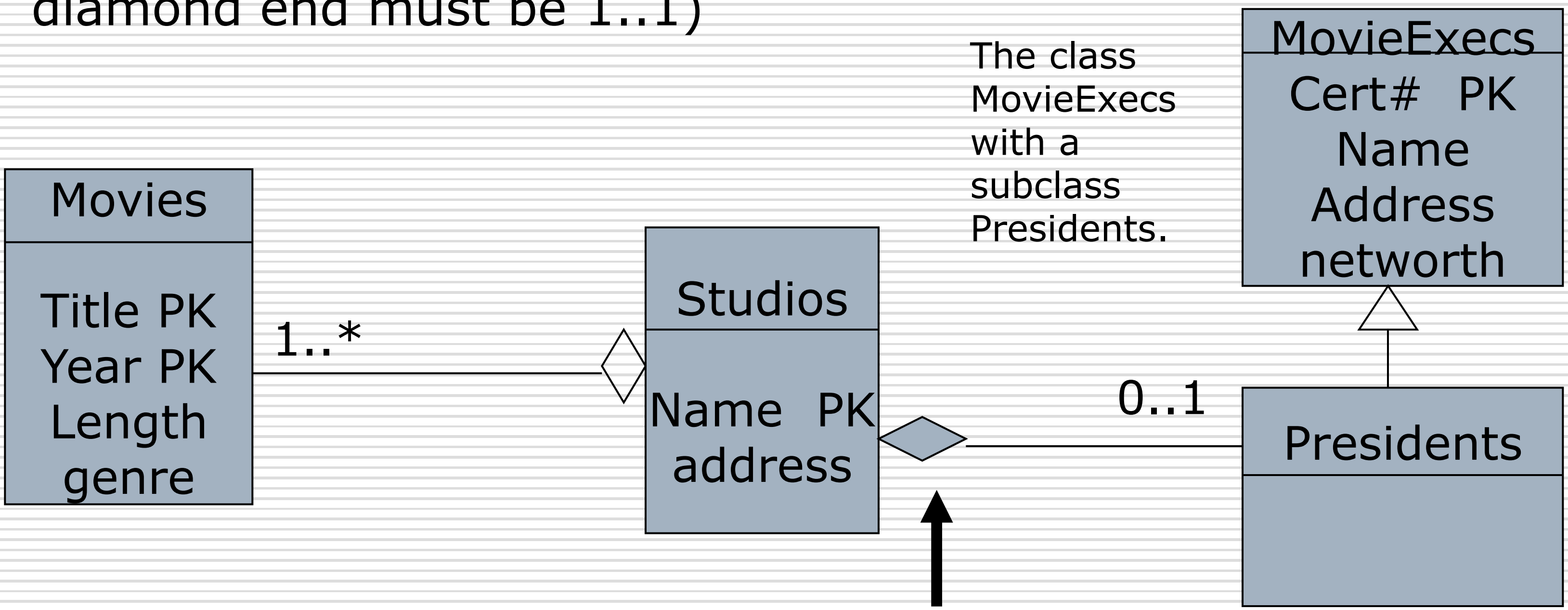
- Relationships with implication that the objects on one side are "owned by" or are part of objects on the other side.
- Represented by a diamond at the end of the connecting line, at the "owner" side.
- Implication that in a relational schema, owned objects are part of owner tuples.

Compositions

- Like aggregations, but with the implication that every object is definitely owned by one object on the other side.
- Represented by solid diamond at owner.
- Often used for subobjects or structured attributes.

Examples of Aggregation and composition

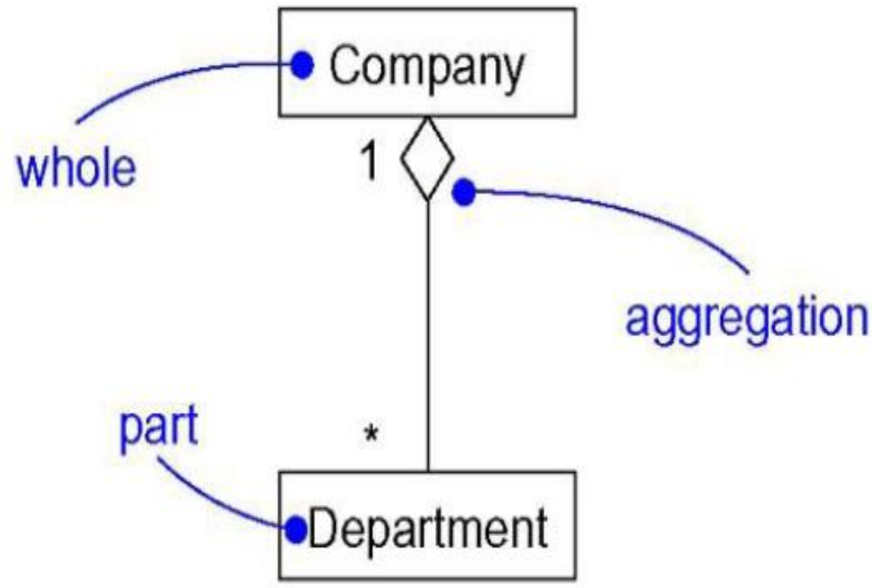
- An aggregation from Movies to Studios (many-one relationship)
- A composition from Presidents to Studios (the lable at the diamond end must be 1..1)

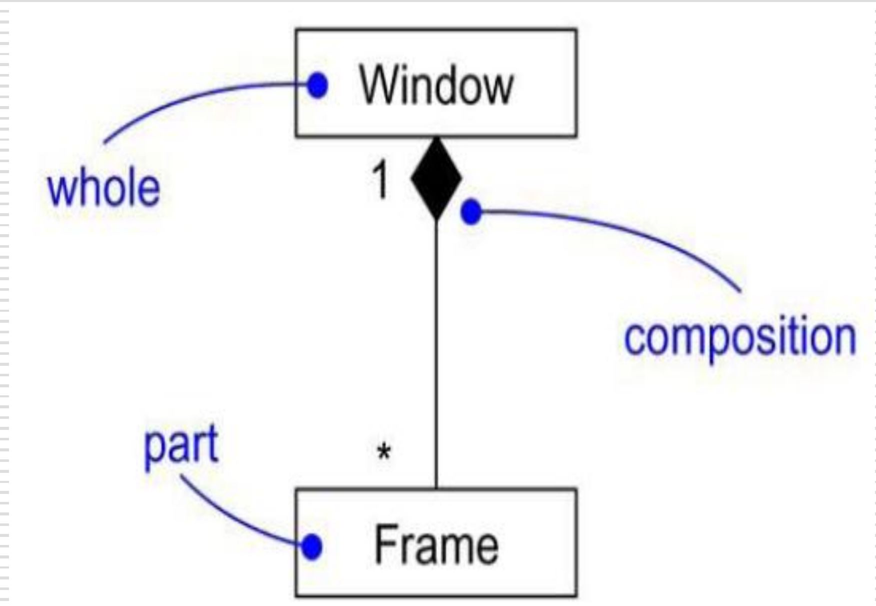


The implication of the composition is that presidents objects will contain a reference to a Studios object and that this reference can not be null

Examples of Aggregation and composition (cont.)

- Both represent Part-whole relationship
- Composition has a strong part-whole relationship, the part and the whole have the same life cycle.



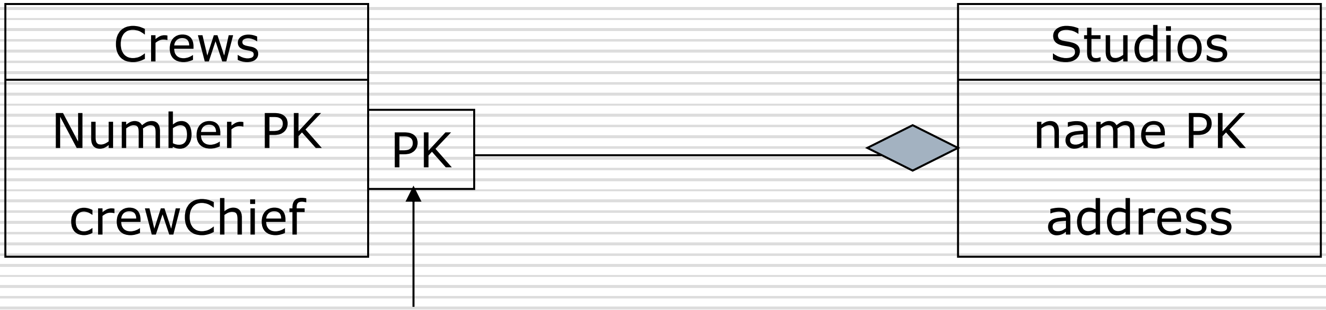


Comparison between UML and E/R model

UML	E/R Model
Class	Entity set
Association	Binary relationship
Association class	Attributes on a relationship
Subclass	Isa hierarchy
Aggregation	Many-one relationship
Composition	Many-one relationship with referential integrity

The UML analog of weak entity sets

- Each object has its own object-identity. -- distinguish two objects even if they have the same values for each of their attributes and other properties.



Weak class Crews supported by a composition and the class Studios

Conversion to Relations

- We can use any of the three strategies outlined for E/R to convert a class and its subclasses to relations.
  1. E/R-style: each subclass' relation stores only its own attributes, plus key.
  2. OO-style: relations store attributes of subclass and all superclasses.
  3. Nulls: One relation, with NULL's as needed.

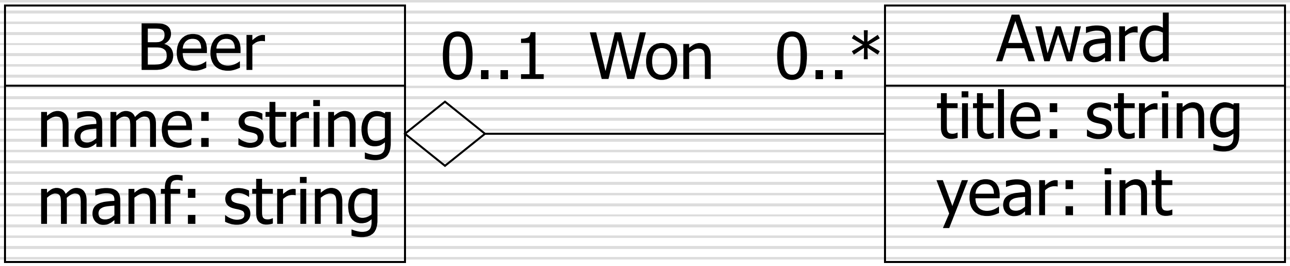
From UML subclass → relations

- If a hierarchy is disjoint at every level, then an object-oriented representation is suggested.
- If the hierarchy is both complete and disjoint at every level, then the task is even simpler.
- If the hierarchy is large and overlapping at some or all levels, then the E/R approach is indicated.

Conversion to Relations (cont.)

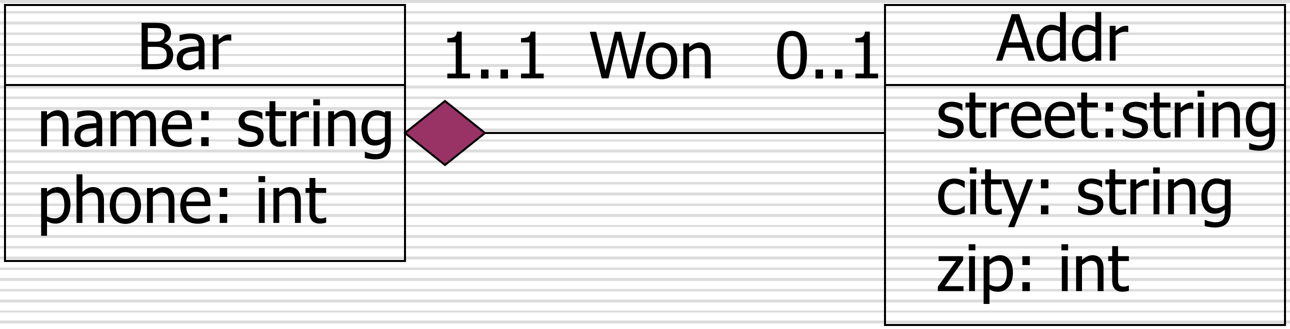
- Classes to Relations.
- Associations to Relations.
- ✓ Aggregations and compositions are types of many-one associations. Construct no relations for them.

From Aggregation to relations



We could store the awards of a beer with the beer tuple. It requires an object-relational or nested-relation model for tables, since there is no limit to the number of awards a beer can win.

From Composition to Relations



- Since a bar has at most one address, it is quite feasible to add the street, city, and zip attributes of Addr to the Bars relation.
- In object-relational databases, Addr can be one attribute of Bars, with structure.

Relationship Comparison between models

- E/R model: many-to-many relationships, multiway relationship, relationship can have an attribute
- UML: many-to-many relationships, relationship can have an attribute
- ODL: many-to-many relationships, relationship has not attributes, with inverse relationship.

Summary

- The E/R model (subclass, weak entity sets)
- UML model
- ODL (keys, relationships, type system)
- Transfer E/R to relational model (Isa hierarchies)
- Transfer UML to relations
- Transfer ODL to relations

Classroom Exercises of chapter 4

- Exercise 4.2.1 (design)
- Exercise 4.4.1
- Exercise 4.4.2

- Homework
- Read subClass section