Database
Intro-DataBase

Université Grenoble Alpes

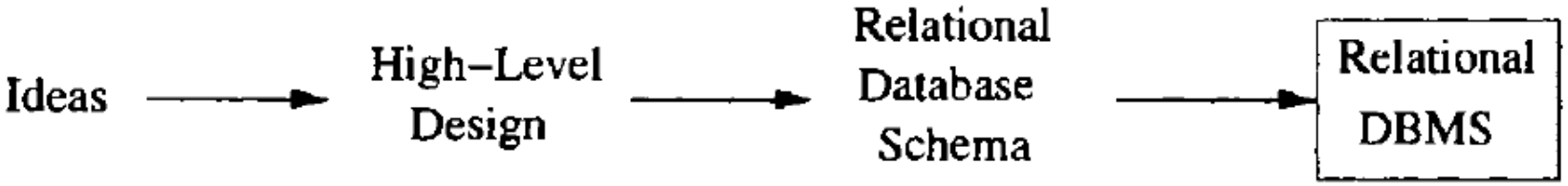
01.03.2023

bahareh.afshinpour@univ-grenoble-alpes.fr

Main reference:
A First Course in Database Systems (and associated material) by
J. Ullman and J. Widom, Prentice-Hall

Recall

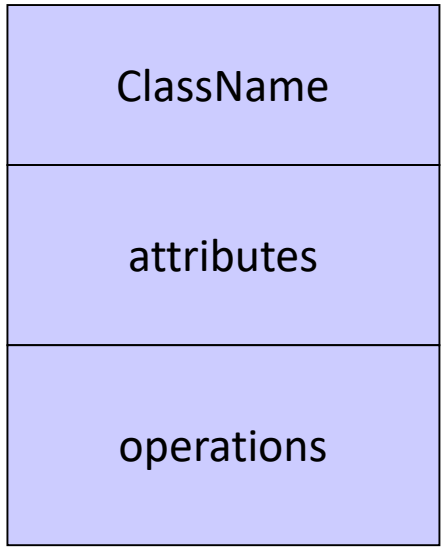
- In practice, it is often easier to start with a higher-level model and then convert the design to the relational model.



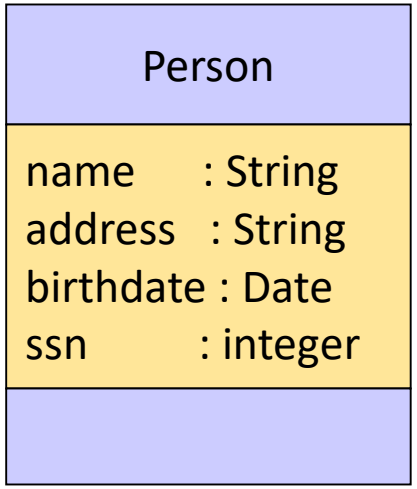
The database modeling and implementation process

- There are several options for the notation in which the design is expressed.
  - Entity-relationship diagram
  - UML (class diagram)
  - ODL(object description language)

Classes



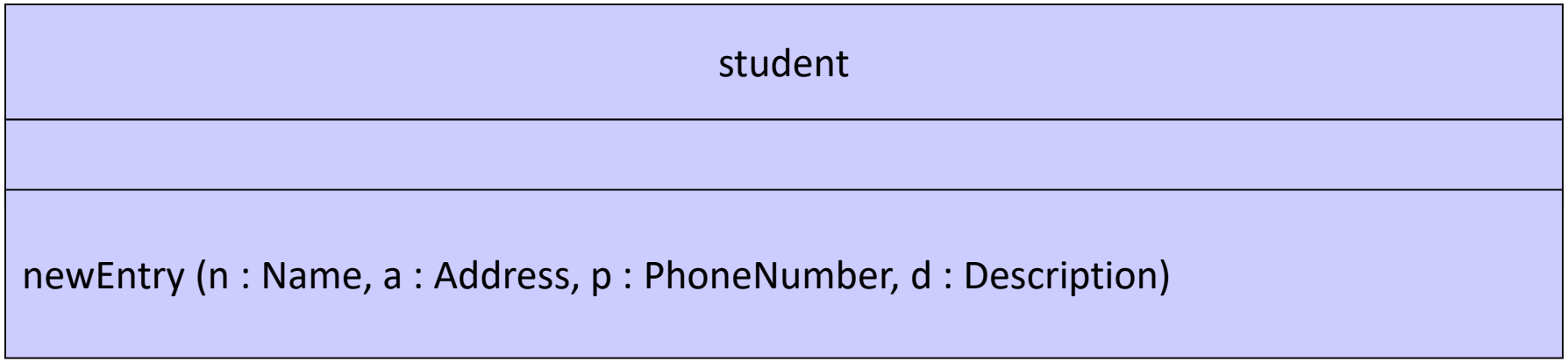
A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.



Attributes can be:

+ public
# protected
- private

Class Operations

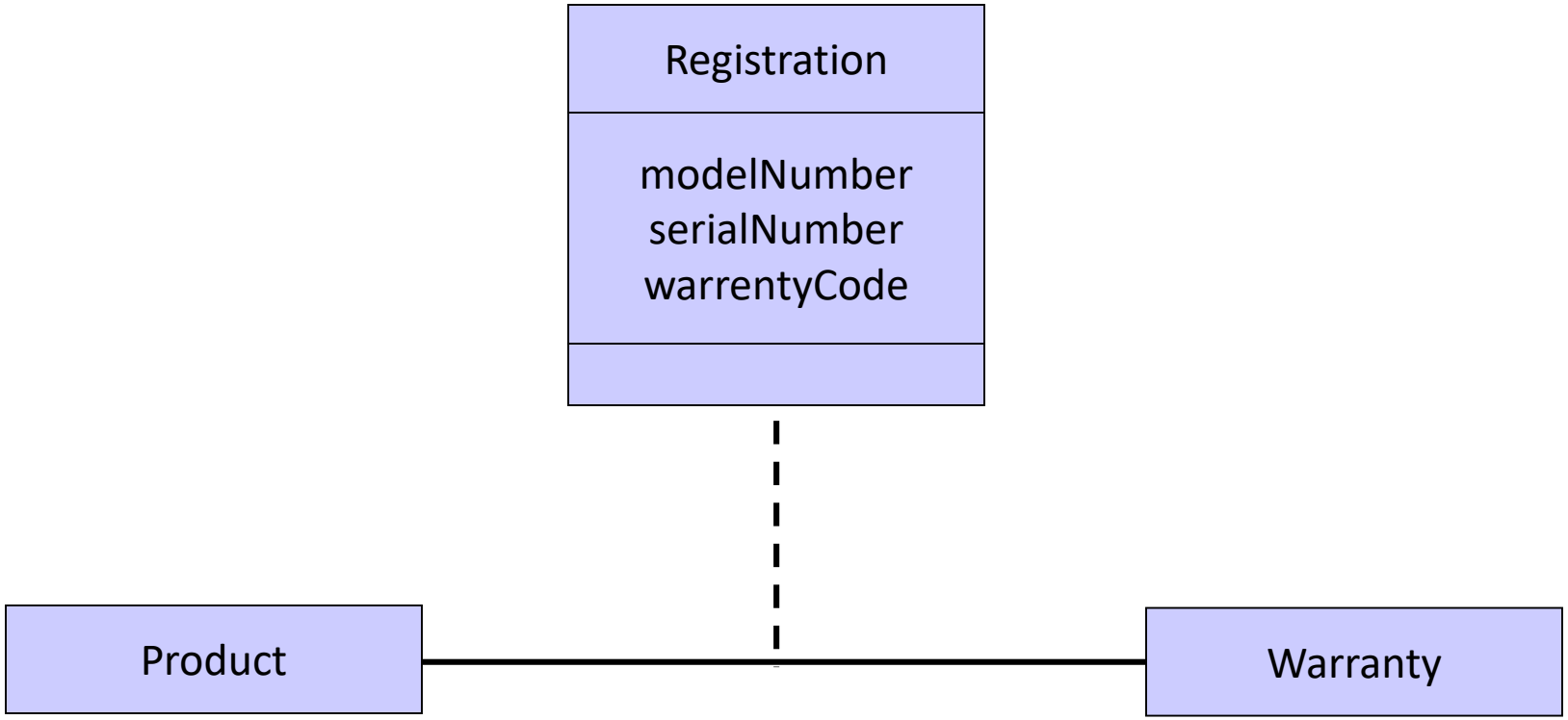


Operations describe the class behavior
and appear in the third compartment.

You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

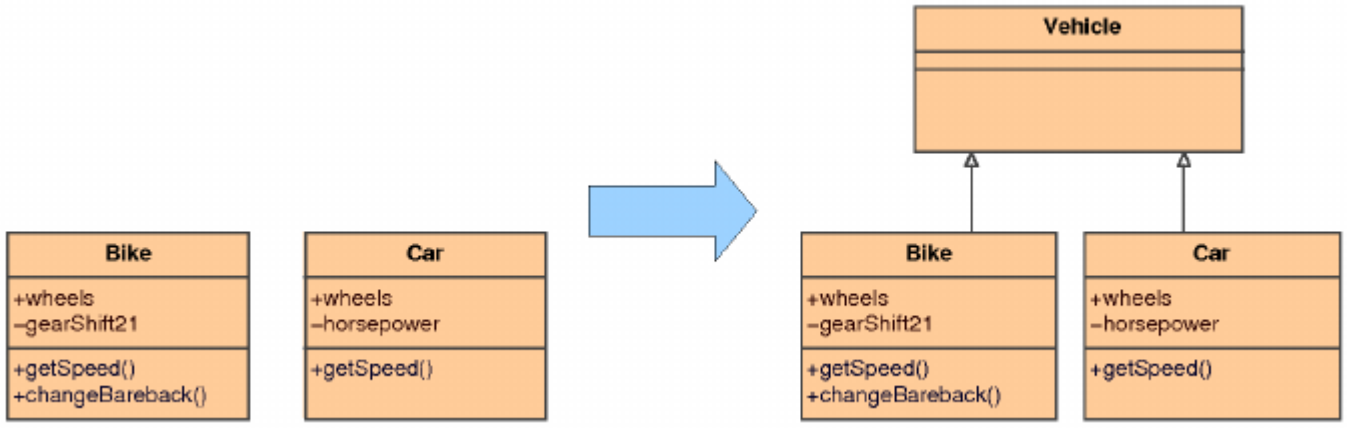
Association Relationships

Associations can also be objects themselves, called *link classes* or an *association classes*.

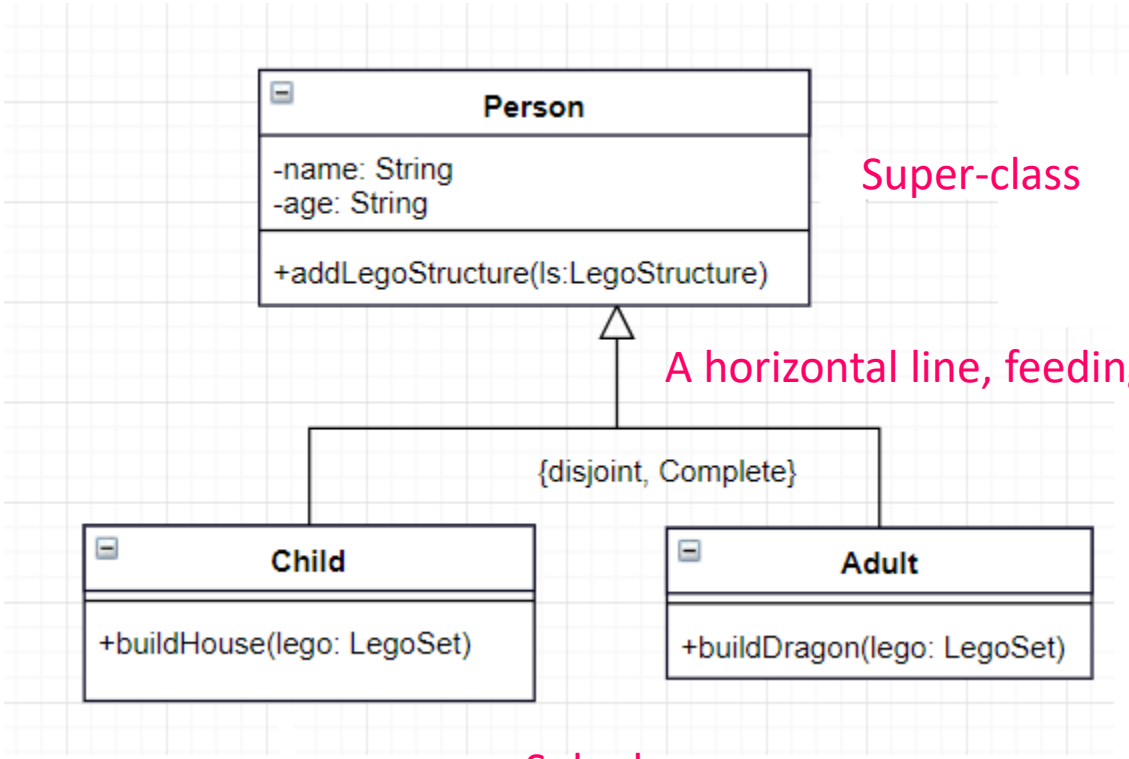


Subclasses in UML

- Subclasses are presented by rectangles, like any class.
- We assume a sub-class inherits the properties(attributes and associations) from its superclass.



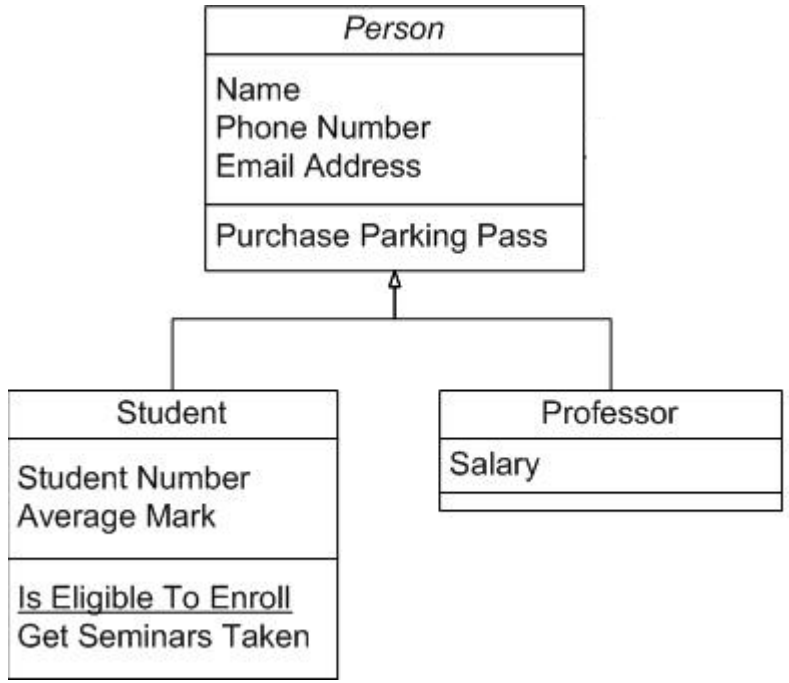
Subclasses in UML



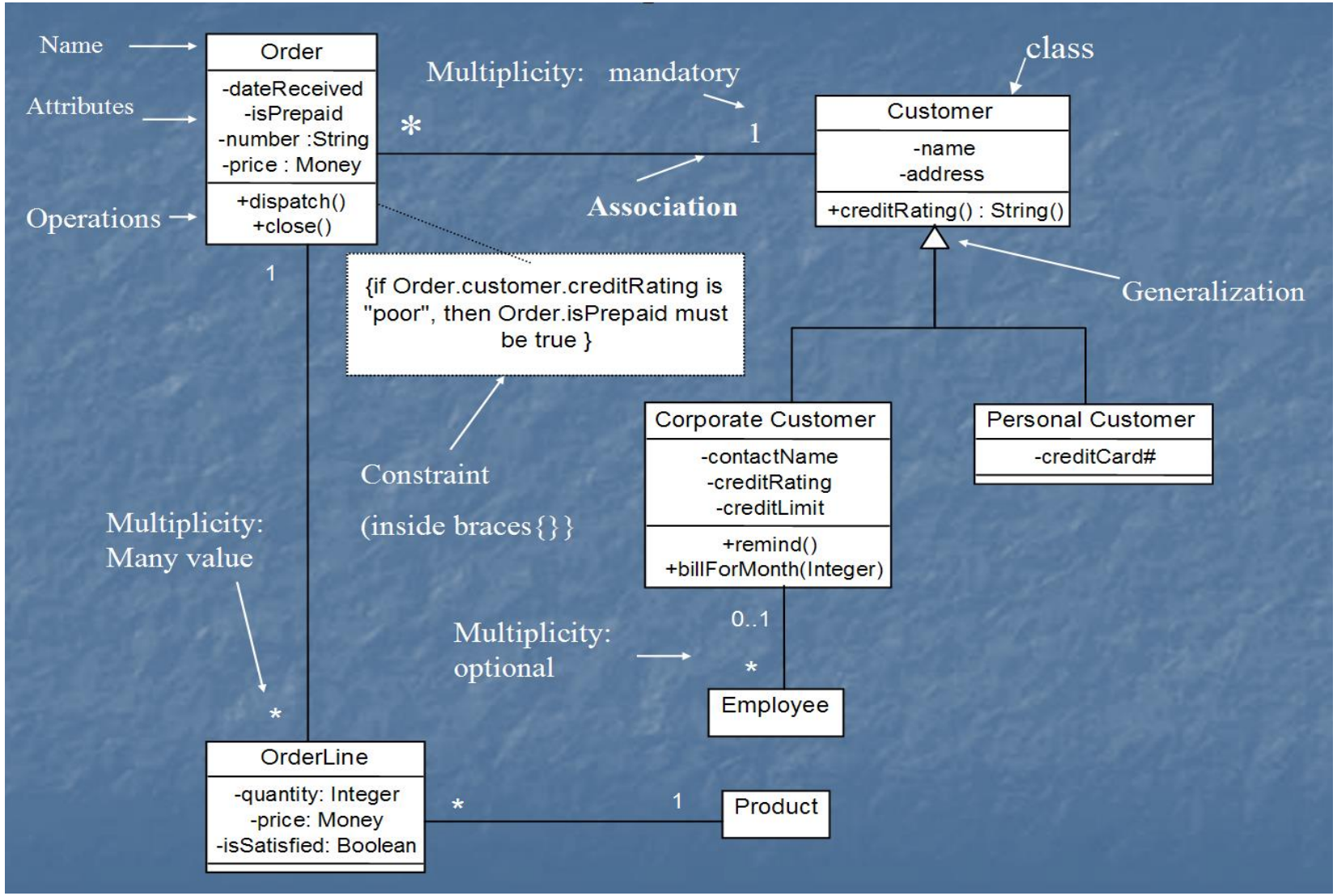
Super-class

A horizontal line, feeding into the arrow

Sub-class

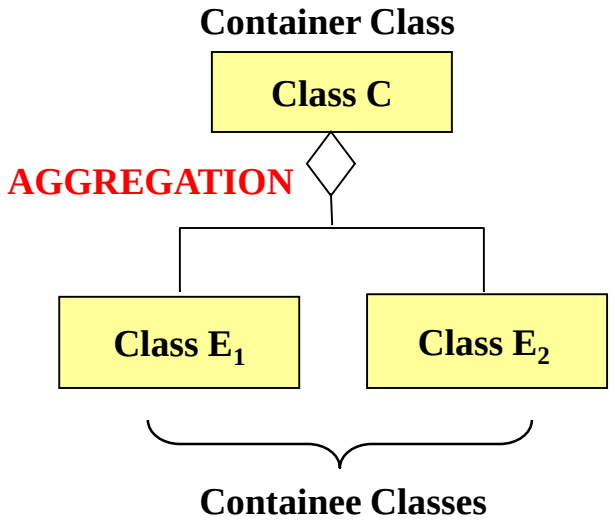


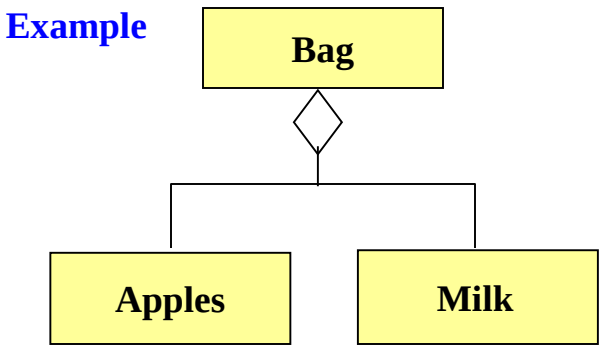
Sub-class may have its own attributes and additional, association


Name
Order
-dateReceived
-isPrepaid
-number :String
-price : Money
+dispatch()
+close()
Attributes
Operations
Multiplicity: mandatory
*
1
Association
class
Customer
-name
-address
+creditRating() : String()
Generalization
1
{if Order.customer.creditRating is "poor", then Order.isPrepaid must be true }
Constraint
(inside braces{})
Corporate Customer
-contactName
-creditRating
-creditLimit
+remind()
+billForMonth(Integer)
Personal Customer
-creditCard#
Multiplicity: Many value
*
Multiplicity: optional
0..1
*
Employee
OrderLine
-quantity: Integer
-price: Money
-isSatisfied: Boolean
*
1
Product

Aggregations

- expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

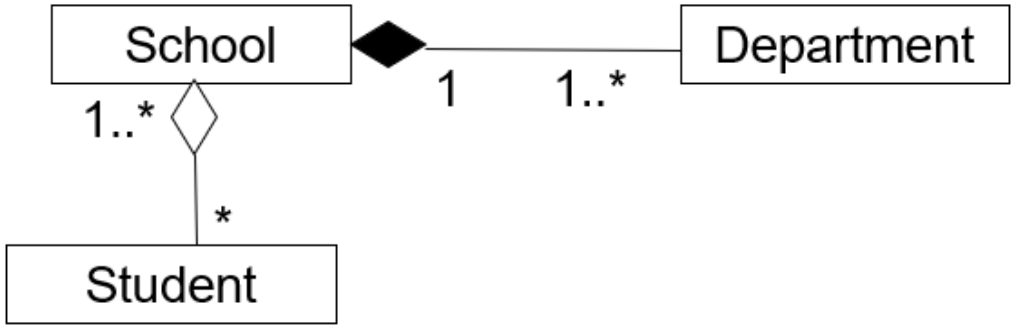




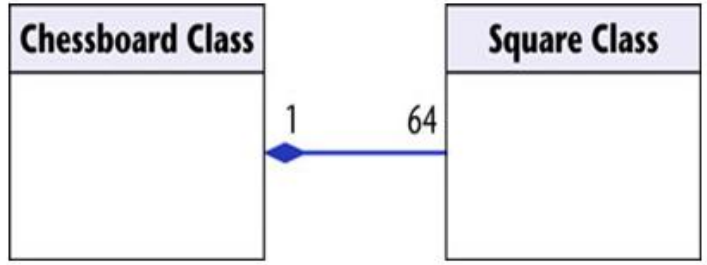
[From Dr.David A. Workman]

Composition

- Stronger relationship
  - ✓ One can not exist without the other
  - ✓ If the school folds, students live on
    - ♦ but the departments go away with the school

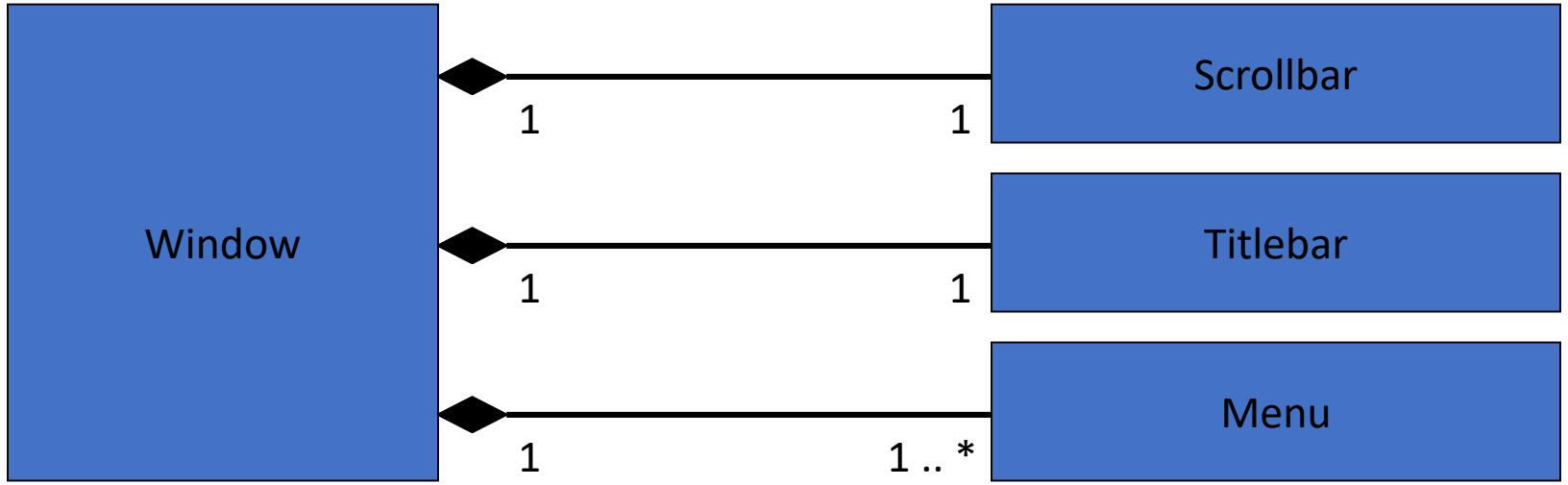


However, it is also possible in UML, to use **composition** as we used supporting relationships for **weak entity sets** in the E/R model.



The McGraw-Hill Companies, 2005 Figure 16.7

- ✓ Model aggregation or composition? When in doubt, use association (just a simple line)


Window
Scrollbar
Titlebar
Menu
1
1
1
1
1
1 .. *

From UML Diagram to Relations

- **Class to relations**
  - For each class, create a relation whose name is the name of the class
  - And whose attributes are the attributes of the class.
- **Associations to Relations**
  - For each association, create a relation with the name of that association
  - The attributes of the relation are the key attributes of the two connected classes (Rename if necessary).
  - If there is an association class attached to the association, include the attributes of the association class among the attributes of the relation.

Example 4.42: Consider the UML diagram of Fig. 4.36. For the three classes we create relations:

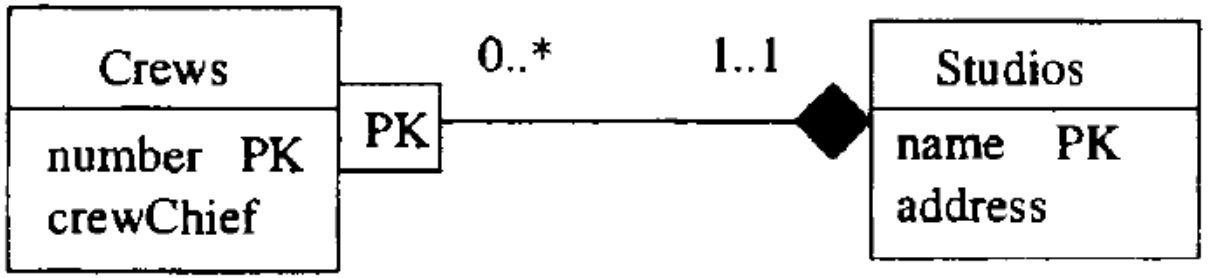
```
Movies(title, year, length genre)
Stars(name, address)
Studios(name, address)
```

For the two associations, we create relations

```
Stars-In(movieTitle, movieYear, starName)
Owns(movieTitle, movieYear, studioName)
```

```
Stars-In(movieTitle, movieYear, starName, salary, residuals)
```

Examples



- Box labeled “PK” indicates that this composition provides part of the key for crews.
- The relation for class crews includes not only its own attribute **number**, but the key for class at the end of the composition, which is studios(name).

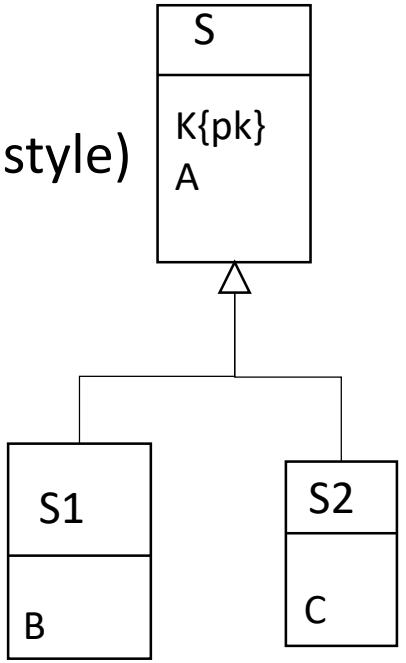
The relations for Example 4.44 are thus:

```
Studios(name, address)
Crews(number, crewChief, studioName)
```

As before, we renamed the attribute *name* of *Studios* in the `Crews` relation, for clarity.

Converting sub-classes

- Three approaches to convert entity sub-classes in relations
  - Subclass relations contain superclass key + specialized attrs. (“UML” style)
  - Subclass relations contain all attributes (“OO” Style)
  - One relation containing all superclass + subclass attrs.



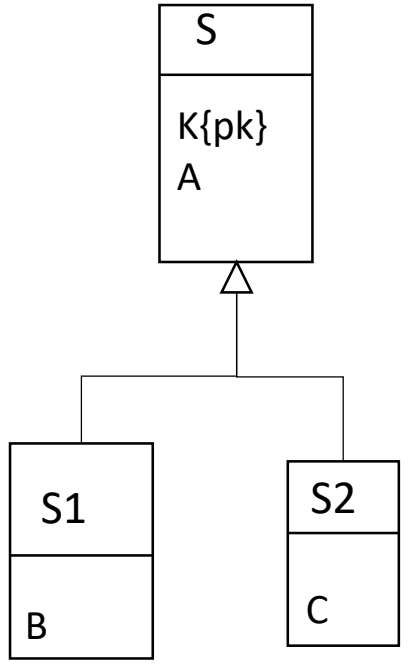
- Pros/cons depend on:
  - the frequent queries…
  - data characteristics
  - sub-classes type (complete/partial; disjoint/overlapping)

Converting entity sub-classes: “UML” style

- Create a relation for the “root” class (as usual)
  - It’s key k is the identifier of the class

- For each sub-class create a relation with the key attributes (k) + **its own specific attributes**

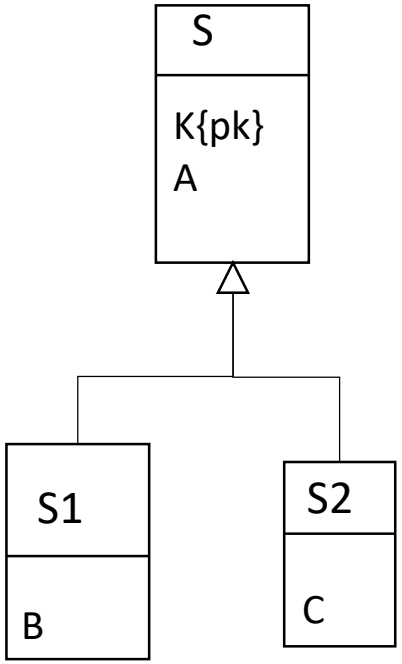
  S(K,A) S1(K,B) S2(K,C)



Converting entity sub-types: “OO” style

- Create a relation for each class and for each sub-class with all its attributes (**own+inherited**)
  - The key is based on the identifier of the “root” entity

    S(K,A) S1(K,A, B) S2(K,A,C)



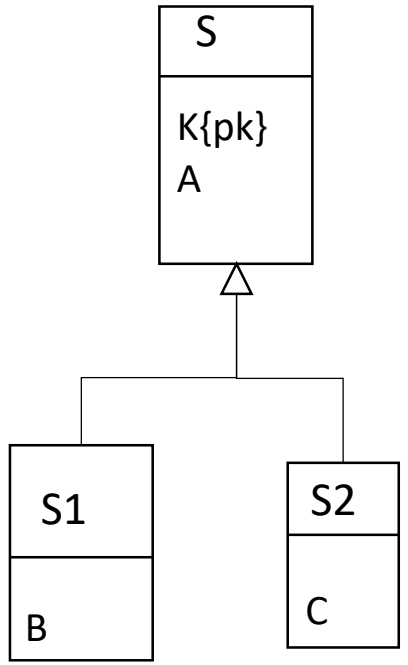
Converting sub-classes: attributes and null values

- Create one single relation with all the attributes of the class hierarchy

  S(K,A, B, C)

- Instances have null in attributes that don't belong to them

- Specific attributes can be used to reflect sub-classes



Object Definition Language

- ODL is as a text-based language for specifying the structure of databases in object-oriented terms.
- Like UML, the class is the central concept in ODL.

A declaration of a class in ODL, in its simplest form, is:

```
class <name> {
    <list of properties>
};
```

Attributes in ODL

- In ODL, attributes need not be of simple types such as integers
- An attribute is represented in the declaration for its class by the keyword attribute, the type, and the name of attribute.

```
class Movie {
   attribute string title;
   attribute integer year;
   attribute integer length;
   attribute enum Genres
          {drama, comedy, sciFi, teen} genre;
};
```

Here genres is enumerated type (list of symbolic constants).
The four values that genre is allowed to take are drama, comedy, …..

Attributes in ODL

- Atribute Address has a type that is a record structure
- The name of this structure is Addr. It consists of two fields: street and city

```
class Star {
    attribute string name;
    attribute Struct Addr
        {string street, string city} address;
};
```

Relationships in ODL

- An ODL relationship is declared inside a class declaration by the keyword relationship, a type, and the name of the relationship.

- For example, the best way to represent the connection between the Movie and Star classes is with a relationship.
- We add this line in the declaration of class Movie.

```
relationship Set<Star> stars;
```

Multiplicity of relationships

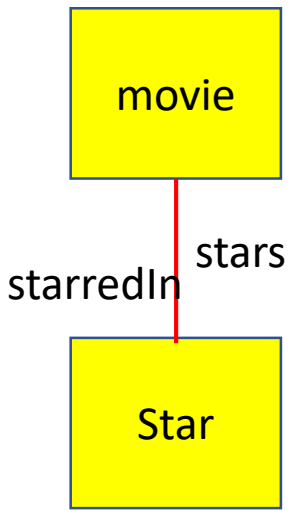
- If we have **many-many** relationships between classes C and D
  - Set<D>, Set <C>
- If the relationship is **many-one** from C to D,
  - The type of the relationship in C is just D
  - while the type of the relationship in D is set<C>.
- If the relationship is **one-one**,
  - the type of the relationship in C is just D
  - and in D it is just C.

```
1)  class Movie {
2)     attribute string title;
3)     attribute integer year;
4)     attribute integer length;
5)     attribute enum Genres
              {drama, comedy, sciFi, teen} genre;
6)     relationship Set<Star> stars
                  inverse Star::starredIn;
7)     relationship Studio ownedBy
                  inverse Studio::owns;
    };

8)  class Star {
9)     attribute string name;
10)    attribute Struct Addr
           {string street, string city} address;
11)    relationship Set<Movie> starredIn
                  inverse Movie::stars;
    };

12) class Studio {
13)    attribute string name;
14)    attribute Star::Addr address;
15)    relationship Set<Movie> owns
                  inverse Movie::ownedBy;
    };
```

Many-many relationships between Star and movie



Since the type of OwnedBy is Studio, while the type of owns is Set<Movie>, we see that this pair od inverse relationship is many-one from Movie to studio.

Declaring Keys in ODL

- The declaration of a key or keys for a class is optional.
  - ODL assumes that all objects have an object-identity

```
class Movie (key (title, year)) {
```

Subclasses in ODL

- Class C to be a subclass of another class D
  - Follow the name C in its declaration with the keyword **extends** and the name D
  - Then class C inherits all the properties of D and may have additional properties of its own.

```
class MurderMystery extends Movie {
   attribute string weapon;
};
```

From ODL Design to relational Design

- Page 193-196

Algebric and Logical query Languages

Chapter 5

Programming

- We now switch our attention from modeling to programming for relational databases.
- We have two abstract programming language
  - Relational algebra(chapter 2)
  - Logic-based

- We extend the algebra so it can handle several more operations than were described previously.

Relational Algebra Operations

Chapter 2 presented the classical relational algebra.

- The usual set operations: union, intersection, difference

- Operations that remove parts of relations:
    - selection, projection
- Operations that combine tuples from two relations:
    - Cartesian product, join
- Since each operation returns a relation,
    - operations can be *composed*!

Projection

- The Projection operator applied to a relation R, produces a new relation with a subset of R's columns.

title	*year*	*length*	*genre*	*studioName*	*producerC#*
Star Wars	1977	124	sciFi	Fox	12345
Galaxy Quest	1999	104	comedy	DreamWorks	67890
Wayne's World	1992	95	comedy	Paramount	99999

The relation Movies

$\pi_{title,year,length}(\text{Movies})$

title	*year*	*length*
Star Wars	1977	124
Galaxy Quest	1999	104
Wayne's World	1992	95

$\pi_{genre}(\text{Movies})$

genre
sciFi
comedy

Duplicate tuples are eliminated

Selection

- The selection operator applied to a relation R, produces a new relation with a **subset of R's tuples**.
- The tuples in the resulting relation are those that satisfy some condition C that involves the attributes R.

title	*year*	*length*	*genre*	*studioName*	*producerC#*
Star Wars	1977	124	sciFi	Fox	12345
Galaxy Quest	1999	104	comedy	DreamWorks	67890
Wayne's World	1992	95	comedy	Paramount	99999

The relation Movies

$$\sigma_{length \geq 100}(\text{Movies})$$

title	*year*	*length*	*inColor*	*studioName*	*producerC#*
Star Wars	1977	124	sciFi	Fox	12345
Galaxy Quest	1999	104	comedy	DreamWorks	67890

Cartesian Product (cross-product)

- The Cartesian Product of two sets R and S is the set of pairs that can be formed by choosing the first element from R and the second from S.
- If R and S have some attributes in common, we need to invent a new name for the identical attributes.

Relation R

A	B
1	2
3	4

Relation S

B	C	D
2	5	6
4	7	8
9	10	11

Relation R X S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Bags

In this section, we shall consider relations that are bags (multisets) rather than sets. That is, we shall allow the same tuple to appear more than once in a relation. When relations are bags, there are changes that need to be made to the definition of some relational operations, as we shall see.

5.1.1 Why Bags?

As we mentioned, commercial DBMS's implement relations that are bags, rather than sets. An important motivation for relations as bags is that some relational operations are considerably more efficient if we use the bag model. For example:

1. To take the union of two relations as bags, we simply copy one relation and add to the copy all the tuples of the other relation. There is no need to eliminate duplicate copies of a tuple that happens to be in both relations.

2. When we project relation as sets, we need to compare each projected tuple with all the other projected tuples, to make sure that each projection appears only once. However, if we can accept a bag as the result, then we simply project each tuple and add it to the result; no comparison with other projected tuples is necessary.

Chapter 5 introduced the modifications necessary to treat relations as **bags** of tuples rather that sets.

Union, Intersection, and Difference

- In the bag union $R \cup S$, tuple t appears $n + m$ times.
- In the bag intersection $R \cap S$, tuple t appears $\min(n, m)$ times.
- In the bag difference $R - S$, tuple t appears $\max(0, n - m)$ times. That is, if tuple t appears in R more times than it appears in S, then t appears in $R - S$ the number of times it appears in R, minus the number of times it appears in S. However, if t appears at least as many times in S as it appears in R, then t does not appear at all in $R - S$. Intuitively, occurrences of t in S each "cancel" one occurrence in R.

Example

Example 5.4: Let R be the relation of Fig. 5.1, that is, a bag in which tuple $(1,2)$ appears three times and $(3,4)$ appears once. Let S be the bag

A	B
1	2
3	4
3	4
5	6

R:
(1,2) 3times
(3,4) 1 time

Then the bag union $R \cup S$ is the bag in which $(1,2)$ appears four times (three times for its occurrences in R and once for its occurrence in S); $(3,4)$ appears three times, and $(5,6)$ appears once.

The bag intersection $R \cap S$ is the bag

A	B
1	2
3	4

with one occurrence each of $(1,2)$ and $(3,4)$. That is, $(1,2)$ appears three times in R and once in S, and $\min(3,1) = 1$, so $(1,2)$ appears once in $R \cap S$. Similarly, $(3,4)$ appears $\min(1,2) = 1$ time in $R \cap S$. Tuple $(5,6)$, which appears once in S but zero times in R appears $\min(0,1) = 0$ times in $R \cap S$. In this case, the result happens to be a set, but any set is also a bag.

The bag difference $R - S$ is the bag

A	B
1	2
1	2

To see why, notice that $(1, 2)$ appears three times in R and once in S, so in $R - S$ it appears $\max(0, 3 - 1) = 2$ times. Tuple $(3, 4)$ appears once in R and twice in S, so in $R - S$ it appears $\max(0, 1 - 2) = 0$ times. No other tuple appears in R, so there can be no other tuples in $R - S$.

Selection on Bags

- To apply a selection to a bag, we apply the selection condition to each tuple independently. As always with bags, we do not eliminate duplicate tuples in the result.

Example 5.5: If R is the bag

A	B	C
1	2	5
3	4	6
1	2	7
1	2	7

then the result of the bag-selection $\sigma_{C \geq 6}(R)$ is

A	B	C
3	4	6
1	2	7
1	2	7

Products of Bags

The rule for the Cartesian product of bags is the expected one. Each tuple of one relation is paired with each tuple of the other, regardless of whether it is a duplicate or not. As a result, if a tuple r appears in a relation R m times, and tuple s appears n times in relation S, then in the product $R \times S$, the tuple rs will appear mn times.

Extended Operators of Relational Algebra

SQL have several other operations that have proved quite important in applications.

1-The duplicate-elimination operators

2-Aggregation operators such as SUM . They apply to attributes(columns) of a relation.

3-Grouping of tuples, can partition the tuples into group. Aggregation can then be applied to columns within each group.

4- Extended projection gives additional power to the operation projection.

5-The sorting operator turns a relation into a list of tuples, sorted according to one or more attributes.

6-The outerjoin operator is a variant of the join that avoids dangling tuples.

Duplicate Elimination $\delta(R)$

- Sometimes we need an operator that converts a bag to a set.

Example 5.8: If R is the relation

A	B
1	2
3	4
1	2
1	2

from Fig. 5.1, then $\delta(R)$ is

A	B
1	2
3	4

Aggregation operators (summarize)

- Many operators we can apply to set or bags of numbers or strings.
- They are used to **summarize or aggregate** the values in one column of relation

- For example:
- SUM
- AVG
- MIN and MAX (numerical values and character-string values)
- COUNT

Example

Example 5.9: Consider the relation

A	B
1	2
3	4
1	2
1	2

Some examples of aggregations on the attributes of this relation are:

1. SUM(B) $= 2 + 4 + 2 + 2 = 10$.
2. AVG(A) $= (1 + 3 + 1 + 1)/4 = 1.5$.
3. MIN(A) $= 1$.
4. MAX(B) $= 4$.
5. COUNT(A) $= 4$.

□