Unified Modeling Language (UML)

CS-A1153: Databases

Prof. Nitin Sawhney

Aalto University
School of Science

Acknowledgements

These slides are based in part on presentation materials created by **Kerttu Pollari-Malmi** and **Juha Puustjärvi** in previous years and on the course text book: **A First Course in Database Systems**, Third Edition. Pearson by Jeffery D. Ullman and Jennifer Widom.

Thanks to **Etna Lindy** & **Ville Vuorenmaa** for translating prior lecture slides for this course.

Learning Goals

- Describing databases with the **Unified Modeling Language (UML)**

- Familiarity with fundamental concepts of UML-modeling:
  - class and attribute
  - association
  - association class
  - subclass
  - composition and aggregation

UML Modeling

- *Problem:* what relations should we define for our database?
- Often it is easier to start planning from a higher abstraction level than to dive into a relational model.
- Here we introduce one way of represent things on a higher abstraction level using UML diagrams.
- UML was originally created for designing object-oriented programs. For database design we only use a small portion of the properties and features of UML.
- Other commonly used methods to represent high-level database designs:
  - E/R diagrams (Entity-Relationship)
  - ODL (Object Description Language)

Differences with object-oriented UML modeling

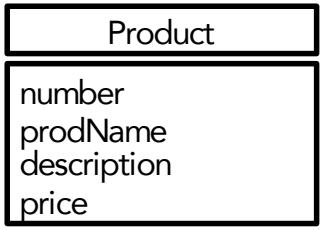
- Unlike object-oriented UML modeling, in database design
  - The classes have no methods
  - Key attributes need to be marked on the diagram
  - Those attributes, that can be retrieved through an association, and won't be written inside the class
  - Types of the attributes are atomic
  - We need to indicate the multiplicity of the association, unless it is *1..1*
  - Lines representing associations often don't have arrowheads

Class in UML

- A class in a UML diagram is roughly equivalent to a class in object-oriented programming – except here classes don't have methods
- Attributes describe the properties of the object

Example of a Class

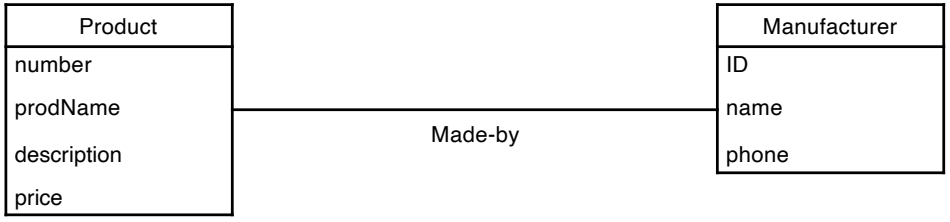
- The class *Product* describes products in an online store.



- For each product available in the store there exists one *Product*-object.
- The attributes of the class are listed under the name of the class. Types of the attributes need to be atomic. Each *Product*-object has its own values for attributes *number*, *prodName*, *description* and *price*.
- When modeling databases with UML diagrams, we don't include any possible actions (methods) for the objects of the class.

Associations

- Associations create a connection between two classes (in a UML diagram no more than two).
- For example between the classes *Product* and *Manufacturer* we might define an association *Made-by*. If a manufacturer *m* has manufactured the product *p*, the object *p* from the class *Product* will be connected to the object *m* of the class *Manufacturer* with association *Made-by*.
- The name of the association is usually written below the line representing the association.



Multiplicity of Association

- On both ends of the line representing the association, it's indicated, how many objects from one class can connect to some other class through an association.

- In this example, there can be 0 or 1 *Manufacturer*-objects for each *Product*-object.
- Each *Manufacturer*-object can connect to arbitrarily many *Product*-objects.
- **Pay attention to which ends the labels are on!**

Multiplicity of Association: Precise Notations

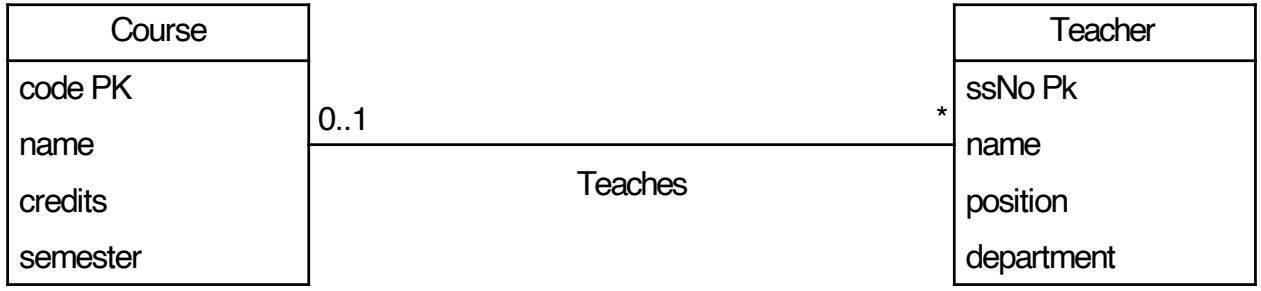
- In an association between classes *A* and *B*, above or below the line, the label m..n **at class *B*'s end** means, that for each object in the class *A* we have at least *m* and at most *n* objects of class *B* connected to it.
- Examples:
  - *0..1* at most 1, but possibly none
  - *0..5* at most 5, but possibly none
  - *1..2* at least 1, at most 2
  - *1..1* exactly 1 (can be labeled with *1*)
  - *1..** at least one 1, but otherwise arbitrarily many
  - *0..** arbitrary amount (can be labeled simply with *).
- When it comes to database modeling, a missing label is equivalent to 1..1.

Multiplicity of Association: Terminology

- *Many-many* association: each object from both classes can have arbitrarily many objects of the other class associated with it.
- *Many-one* association: each object from the first class has at most one object of the other class associated with it.
- *Many-exactly one* association: each object from the first class has exactly one object of the other class associated with it.

Exercise 1

- Which of the following claims hold for the UML diagram of teachers and courses below?



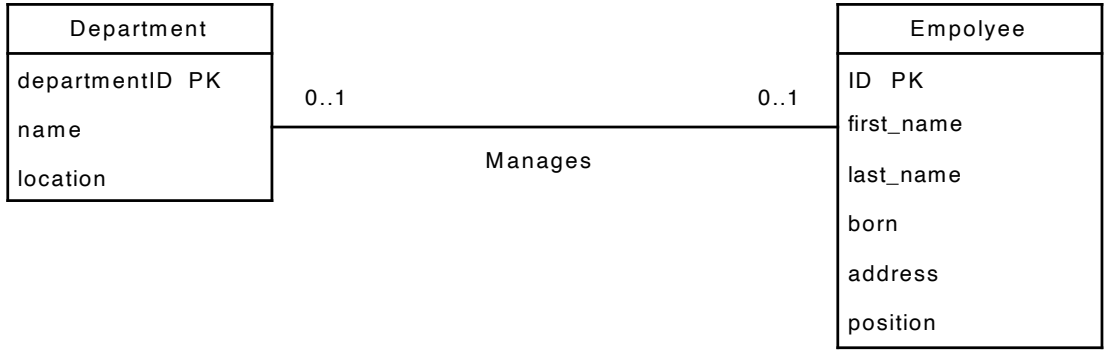
a. One teacher can teach arbitrarily many courses, and one course can have arbitrarily many teachers.
b. One teacher can teach at most one course, but one course can have arbitrarily many teachers.
c. One teacher can teach arbitrarily many courses, but one course can have at most one teacher.
d. One teacher can teach at most one course, and one course can have at most one teacher.

Answer

- In the diagram, at the end of the class *Teacher* we have a star. This means, that each *Course*-object can be associated with arbitrarily many *Teacher*-objects. In other words, each course can have arbitrarily many teachers.
- At the end of class *Course* we have the label 0..1. This means, that there can be 0 or 1 *Course*-objects for each *Teacher*-objects. In other words, each teacher can teach at most one course.
- Option b is correct.

Multiplicity of the association: more terminology

- If the association is many-one with respect to both classes, it's a *one-one* association.
- Example: in a company, each department has at most one manager, and each manager can lead at most one department.



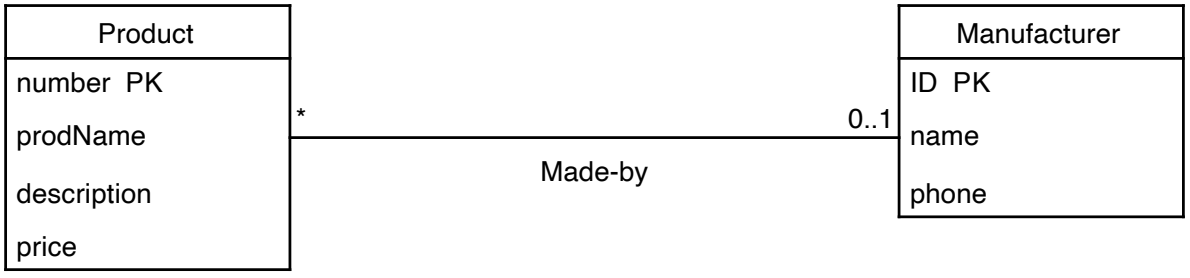
- One-one association is a special case of many-one association. Hence everything that holds for many-one associations, also holds for one-one associations.

Keys

- When designing object-oriented programs, there is no need for keys as in object-oriented programming each object has its own unique identity. However, in databases keys are essential to identify the rows (tuples) of the table. For this reason, the key attributes of the relation should be marked in the UML diagram.
- The key of the class *E* is such an attribute or a set of attributes, that no two distinct objects in class *E* share the values for the attribute, or the value combination for the set of attributes.
- Key attributes are marked with label PK.
- There can be multiple choices for a key, but only one is chosen and marked. If multiple attributes are marked with the PK-label, they together form the set of attributes that is the key of the class.

Example of Marking the Attributes

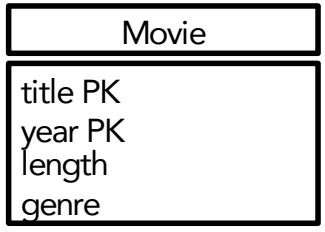
- A product is identified by its product number (attribute *number*), and a manufacturer by its ID (attribute *ID*).



Another Example of Keys

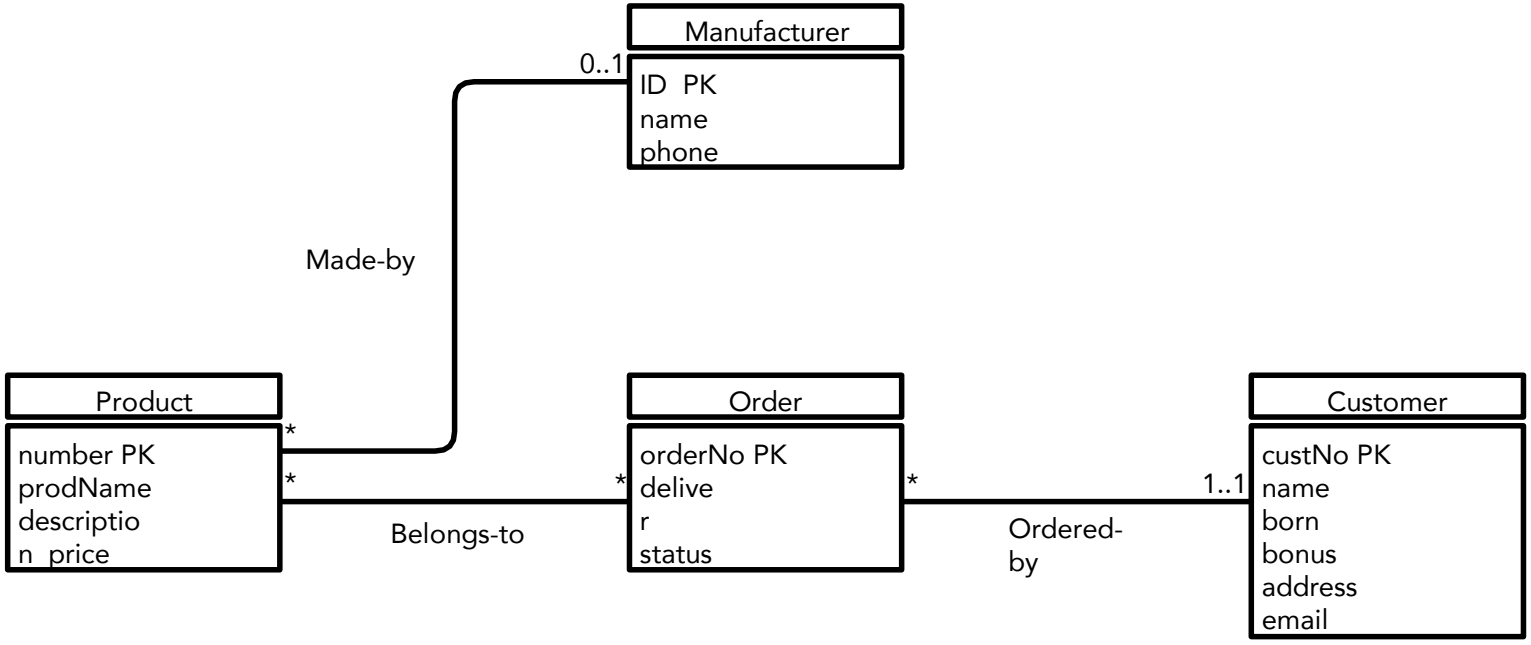
- In the textbook, the keys for the class *Movie* are the name of the movie and the year, together.

The idea is, that no film studio wants to produce a movie with the same name as another movie from a competing studio in the same year. However, it's possible that later someone wants to create a new version of the movie with the same name.



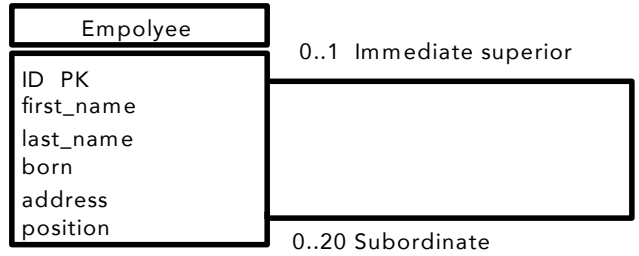
Example Diagram

- We have added the entities *Orders* and *Customers* and the associations *Ordered-by* and *Belongs-to* to our example database for the online store.
- This model allows us to add products with no manufacturer in the database. On the contrary we can't add orders with no customer.



Self-Association

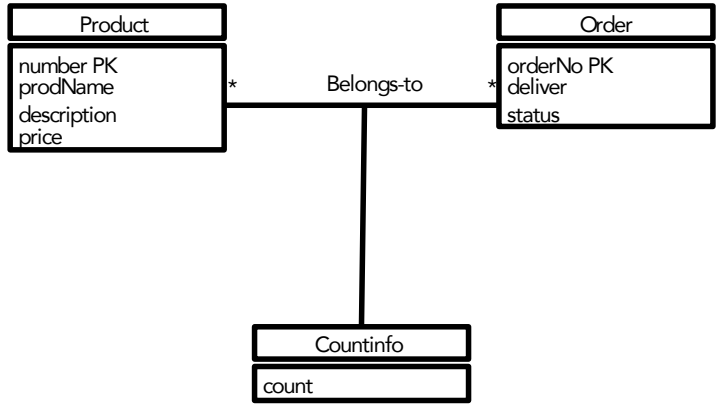
- *Problem:* How can we represent associations where the same class occurs twice?
- Example: let's define, that one employee can be an immediate superior for another employee.



- Both roles are written in the diagram. The multiplicity of the association is indicated at the end corresponding to the roles (here one employee can have at most one immediate superior, and 0–20 subordinates).

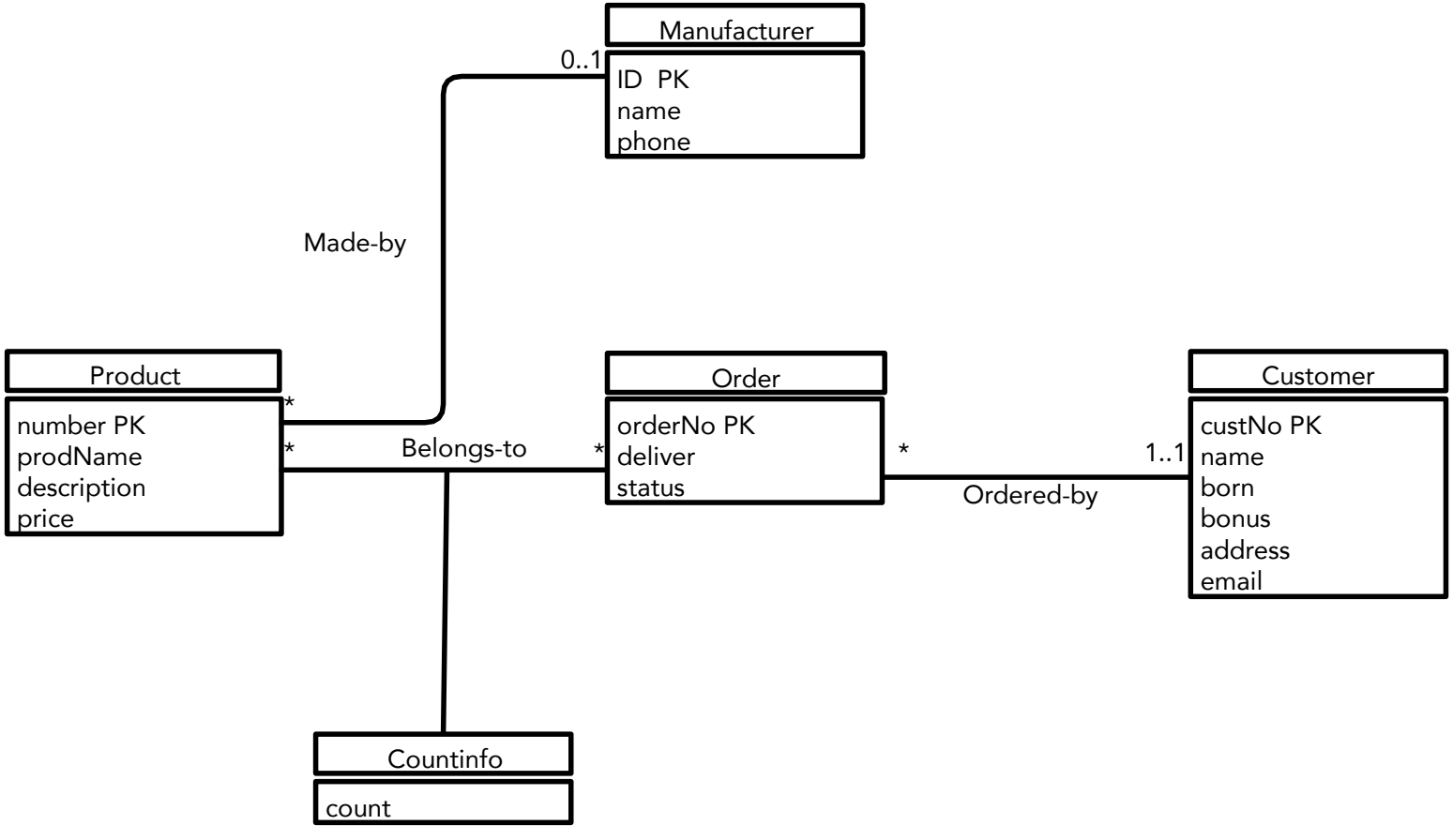
Association Class

- Example: in the web store example, let's specify that a customer can add many items of the same product to one order.
- Where should information about number of items be attached?
- Let's add to association a *Belongs-to* association class, and to it's attribute we'll add information about number of items.



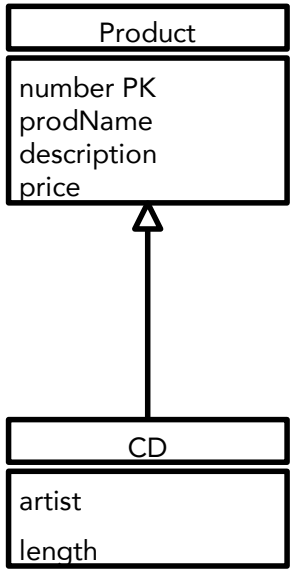
- An association class has no key attributes and multiplicity is never indicated on the line leading to the association class.

Web Store Example with Association Class



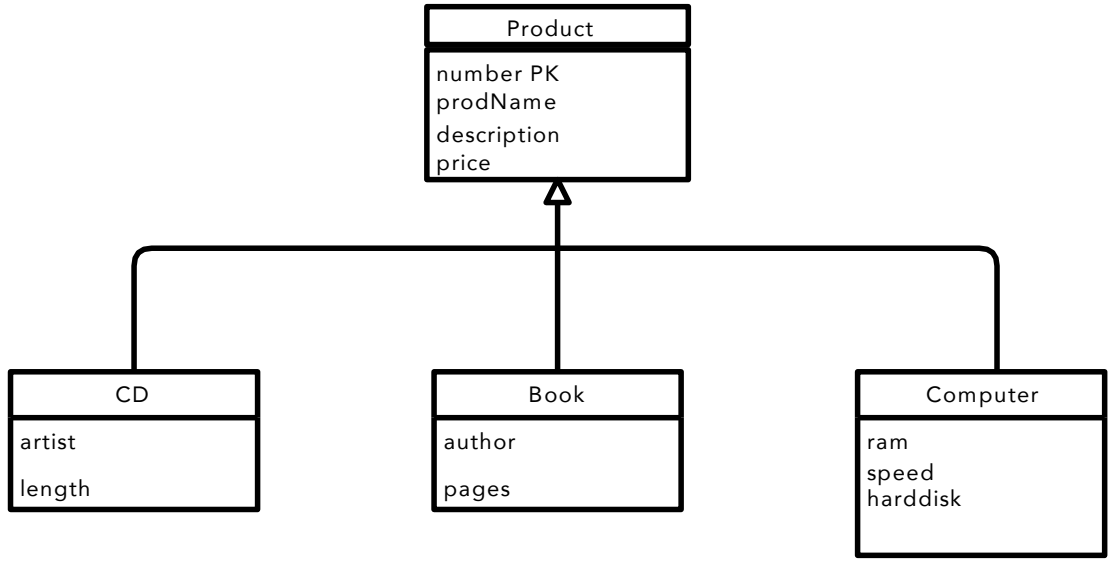
Inheritance and Subclasses

- Only some objects of the class have certain features. For these instances we can define a *subclass*.
- Object of subclass have all the attributes and associations of it's superclass and also the attributes and associations of the subclass.
- Relation from subclass to superclass is marked with a little triangle (pointing to the superclass)



Subclasses in UML-modeling: Example

- In this example the class *Product* has subclasses for different products: *CD*, *Book* and *Computer*.



- Subclasses may have their own associations that other classes of the inheritance hierarchy do not have. For example, class *CD* could have association to class *Track*, which describes one track of the album.

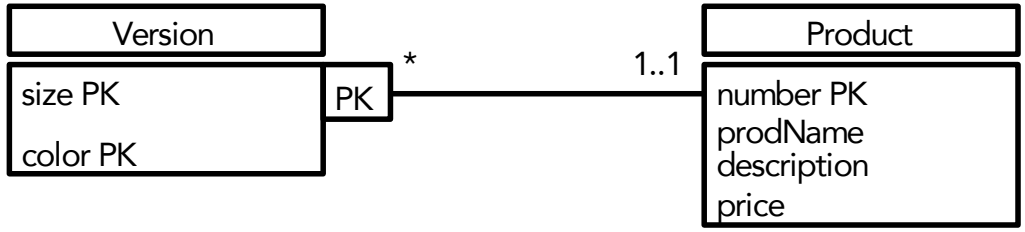
Inheritance and Keys

- Subclass never has it's own key attributes, but all the attributes needed for the key have to be attributes of the superclass.

When Attributes are not enough for the Key

- Let's assume that products of web store can have different versions, for example the same clothing can come in different colors and sizes.
- Let's define a class *Version* to describe different versions of product.
- Attributes of the class are size and color. Even together they are not enough to uniquely identify an object of *Version* (therefore they don't form a key).
- Solution: let's form a key of class *Version* by using attributes of class *Version* and the key attribute of class *Product*.
- The class *Version* is marked then with only it's own attributes.

When Attributes are not enough for the Key: Example

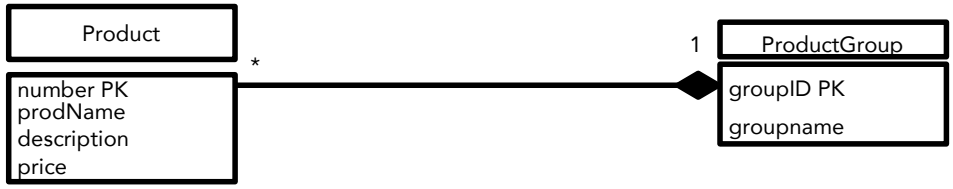


Aggregation

- With aggregation we can tell that an object of some class is formed by objects of another class, for example in the web store there are product groups, that are formed by products.
- Aggregation is marked by an open diamond at the end of that class whose object is formed by objects of another class.

Composition

- Composition is like aggregation, but is a stricter requirement. In composition the object of one class must belong to an object of another class.



UML Modeling

- How can one start UML modeling?
- Simple, but often a good working approach:
  1. Write a description of the database being modeled.
  2. Underline all the nouns.
  3. From nouns, find candidates for classes and attributes.
  4. Some nouns won't become either.
  5. When the classes and attributes are done, think, what kind of relations there might be between the objects of the classes. Make them the associations.

Example: Web Store

- Create a database for a web store that has products and customers.
- Customers can make orders which can include multiple products.
- Products have product number, name, description, price and manufacturer.
- Manufacturers have ID, name and phone number.
- Customers have customer ID, name, year of birth, bonus points, address and email address.
- Every order has a unique order number. Orders also have shipping method, state, products included in the order and customers who made the order.

Example continues: underline attributes

- Create a database for a web store that has products and customers. Customers can make orders which can include multiple products. Products have product number, name, description, price and manufacturer. Manufacturers have ID, name and phone number. Customers have customer ID, name, year of birth, bonus points, address and email address. Every order has unique order number. Orders also have shipping method, state, products included in the order and the customer who made the order.
- "Database" is a common term, which is not related to the modeled object. Therefore it won't become a class or attribute. "Web store" describes the whole system that we are modeling (the whole UML model), so it won't either become a class or attribute either.

Design Principles

What should be considered when making a UML model of a real-life system?

- Faithfulness
- Avoiding Redundancy
- Simplicity Counts
- Choosing the Right Relationships
- Picking the Right Kind of Element

Faithfulness

- Classes and their attributes need to correspond with the real world they are describing.
- For example, multiplicity of association needs to be decided by the fact that the same object in real-life can be associated with one or more objects of other class.
- In a web store the same order can include several products, but the order must have exactly one customer. This is shown in the UML diagram.

Avoiding Redundancy

- One entity should be created only once.
- For example, class *Product* shouldn't have attribute of manufacturer ID, because association *Made-by* already indicates the manufacturer
- Why redundancy is harmful?
  - Repeating same information takes up unnecessary space.
  - Repeating the information may cause problems with database updates.
- Object-oriented programming works differently because from the UML diagram you want to directly access what attributes the classes of the object program have. When designing databases, the situation is different because the classes are not representing directly tables coming into the database.

Striving for Simplicity

- Model should not include additional elements.
- For example, class should not be separated into two classes and an association between them without a good reason.

Figure 11: A poor design with an unnecessary entity set

Choosing the Right Relationships

- Entities can be connected in various ways to relationships.
- Adding to our design every possible relationship is not a good idea.
- Doing so can lead to redundancies and anomalies.

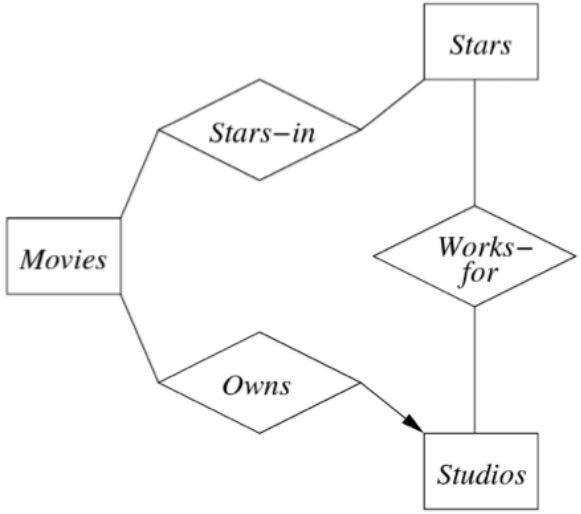
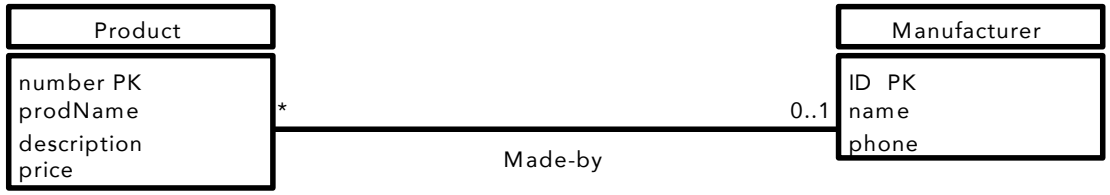


Figure 12: Adding a relationship between *Stars* and *Studios*

Using the Right Elements

- When should some element be described with a class or with an attribute?
- Let's examine the model below. Could class *Manufacturer* and association *Made-by* be replaced by adding additional attributes to the class *Products*?



Using the Right Elements, Continued

- In principle, the class *Products* could have attributes *manufacturerID*, *manufacturerName* and *phone*, which would replace the class *Manufacturer* and the association *Made-by*.
- But then the name and phone number of a certain manufacturer would be repeated in differenct products.
- *Common principle:* if something other than just name or numerical value of some real-world concept is being modeled, it should be modeled as class instead of attribute (however, class can be association class).
- When to use a regular class vs. an association class? Association class is good for situations where the information being described is related exactly to a pair formed by two objects and it does not have its own key.

Using the Right Elements, Continued

- In the web store example *Order* couldn't be an association class, but is has to be regular class, because:
  1. Order is not necessarily a relation between exactly one product and one customer, but one order can include several products.
  2. The class *Order* has it's own key attribute.

Only one of these reasonings would be enough.

UML Modeling using *DbSchema*: Example