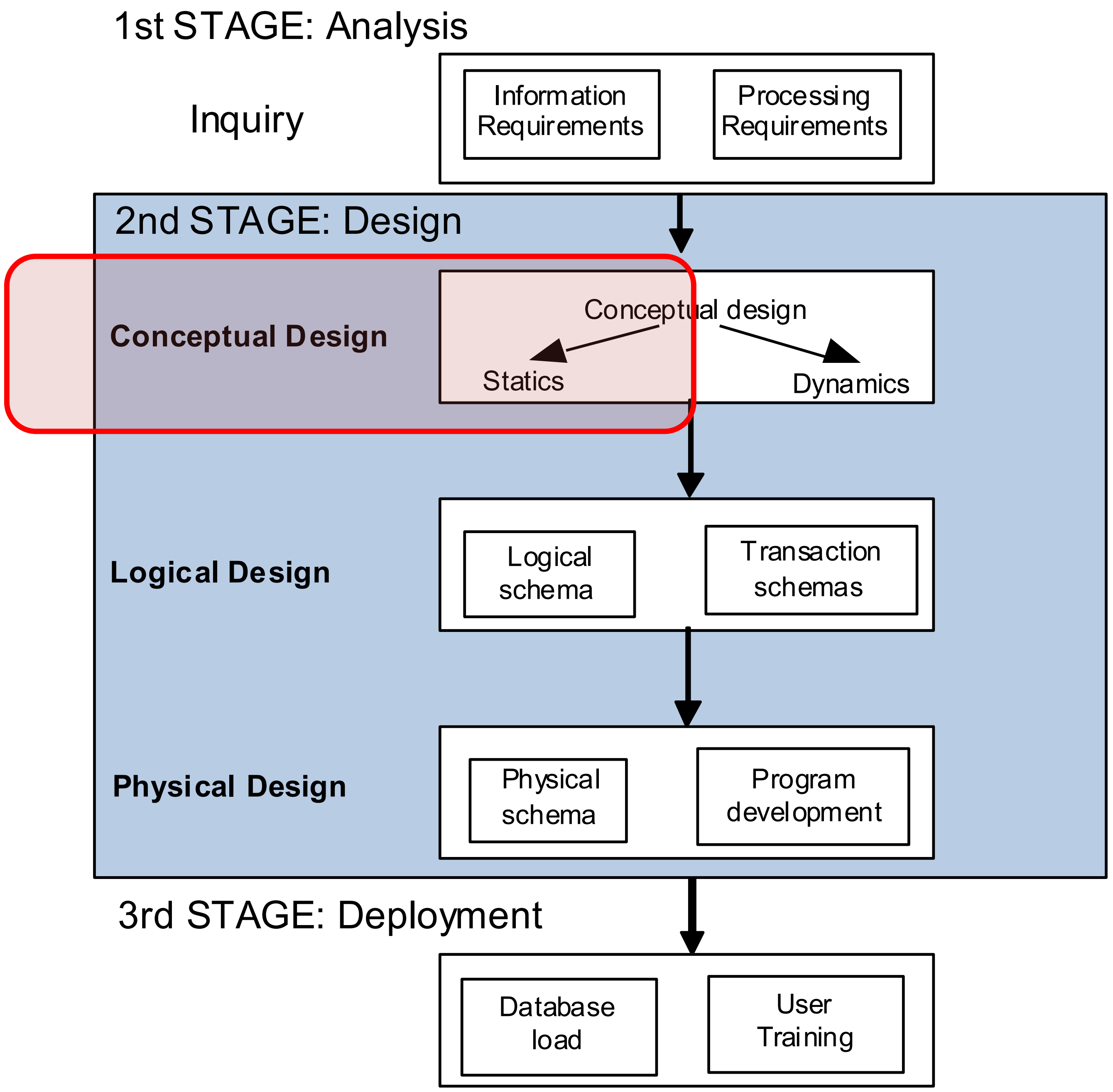
Unit 4: Relational Database Design

4.2. Conceptual Design

1. Introduction



Conceptual Design

Stage of the database design process which aims at "obtaining a representation of reality which captures its static and dynamic properties such that requirements are satisfied. This representation must be a truthful image of the actual world".

For the static conceptual design we will use

UML Class Diagrams

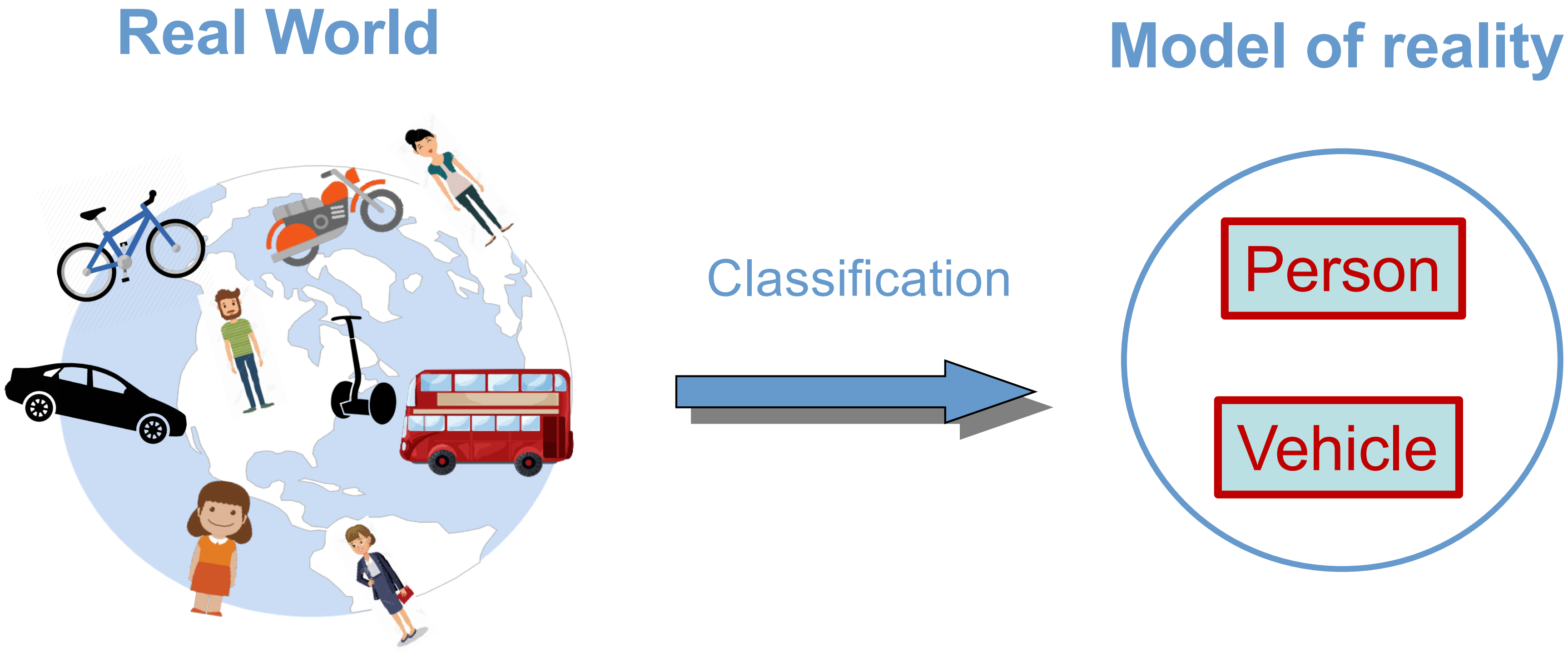
- We will use an enhanced variation of UML (class) diagrams, following *Connolly and Begg* (2014).
- Our enhanced **UML diagrams** will represent:
  - the structures which constitute the content of the information system, and,
  - the constraints which limit the occurrences of the data.
- Some of the elements that we will see:
  - class (entity)
  - attribute
  - association (common relationship)
  - generalization/specialization (other kind of relationship)
  - constraints

4.2. Conceptual Design

2.1. Class

The observation of reality leads to the detection of (physical or abstract) “objects” or “entities”.

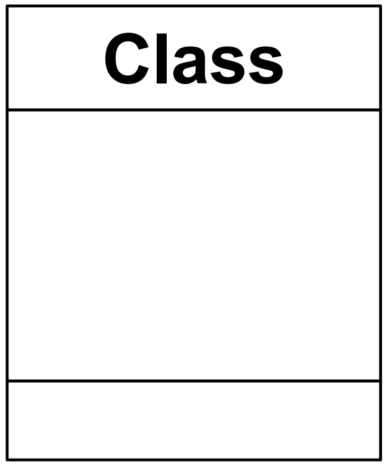
Through **classification** (a very simple abstraction mechanism) we identify the kind of “classes” or “entity types” (types of objects) which are interesting for the organization.



- Object which are of the same kind are represented by a class (or entity)..

- A **class** is a description of a set of objects which share the same properties

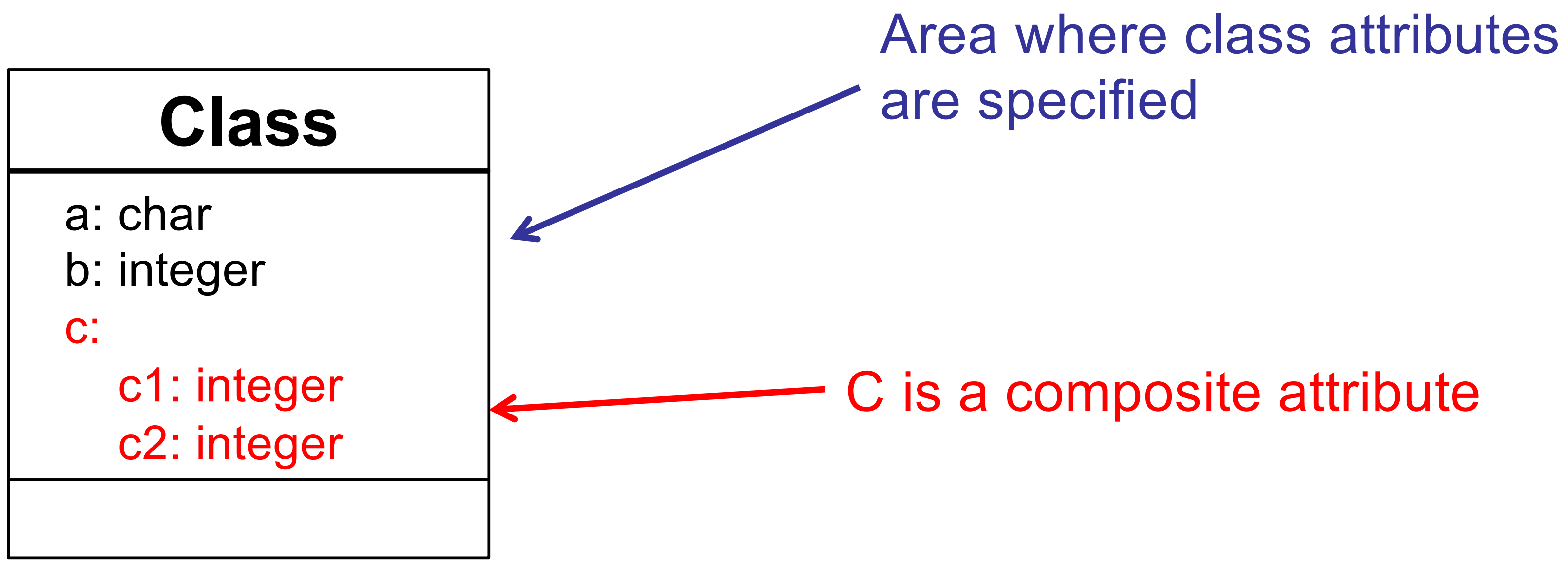
Diagrammatic representation



4.2. Conceptual Design

2.2 Attribute

An **attribute** is a property of a class which is identified by a name, and whose instances can have different values from a given specified set.


Class
a: char
b: integer
c:
c1: integer
c2: integer
Area where class attributes are specified
C is a composite attribute

For each attribute we can specify:

Data type or domain

A predefined datatype which determines the possible values for the attribute:

- The name of a datatype.
- An enumeration of possible values (in parentheses).
- A record (composite attribute).

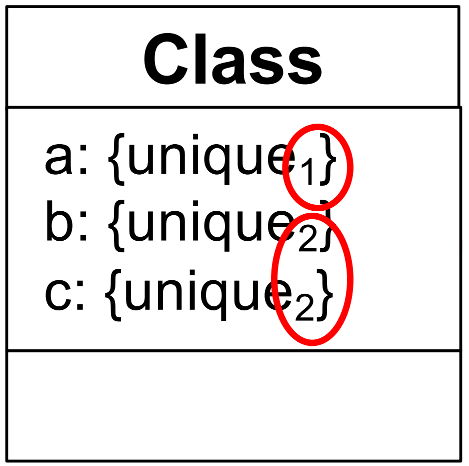
Constraints:

- Uniqueness
- Multiplicity
- Identification

- Derived attributes are special attributes whose value is determined by a rule or formula.

Uniqueness constraints

Different occurrences of a class must take different values for the attribute (or set of attributes) with the uniqueness constraint.



There cannot be two occurrences of this class with

1. The same value for *a*
2. Or with the same values of both *b* and *c*

Cardinality (multiplicity constraint)

Expresses the number of values that each attribute can take for each object in the class.

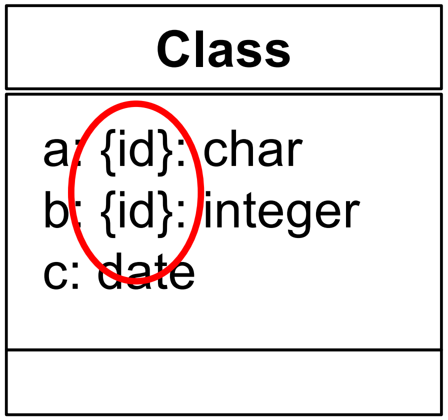
- **{1..1}:** The attribute has exactly one value for each occurrence in the class
- **{0..1}:** The attribute can have no value or just one (this is the default case).
- **{1..*}:** The attribute must have one or more values (but at least one).
- **{0..*}:** The attribute can have no values of any number or values

Identification constraint

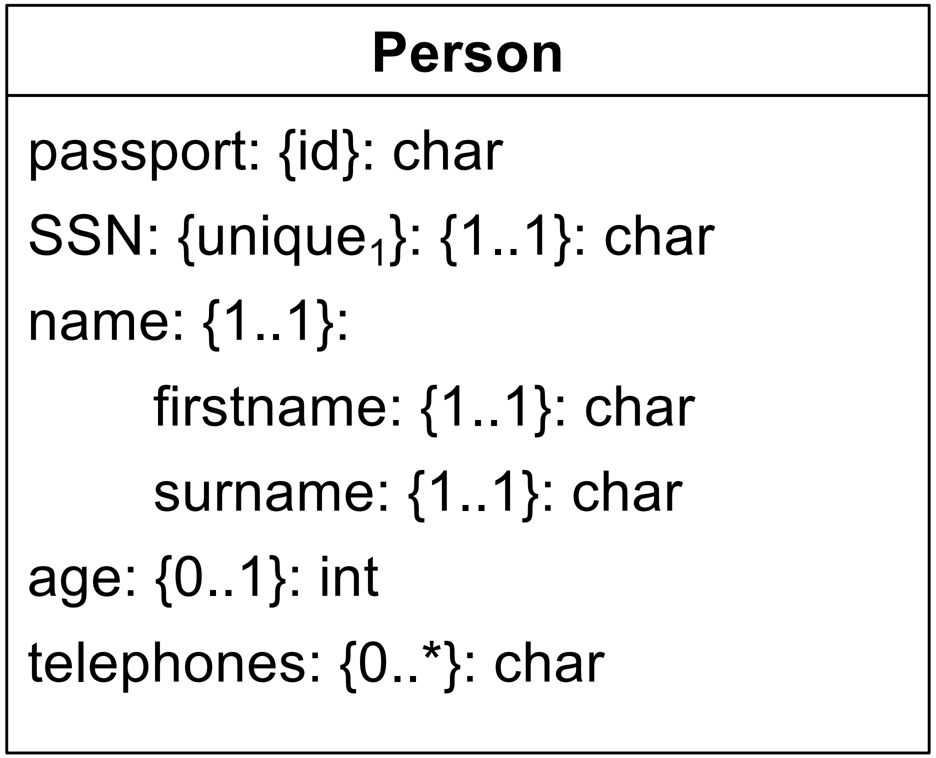
A **identifier** is as set of attributes with uniqueness constraint and multiplicity {1..1}

It allows distinguish any two occurrences of the class.

There is only one identifier per class (but may contain several attributes)



Example



Person

passport: {id}: char

SSN: {unique$_1$}: {1..1}: char

name: {1..1}:

    firstname: {1..1}: char

    surname: {1..1}: char

age: {0..1}: int

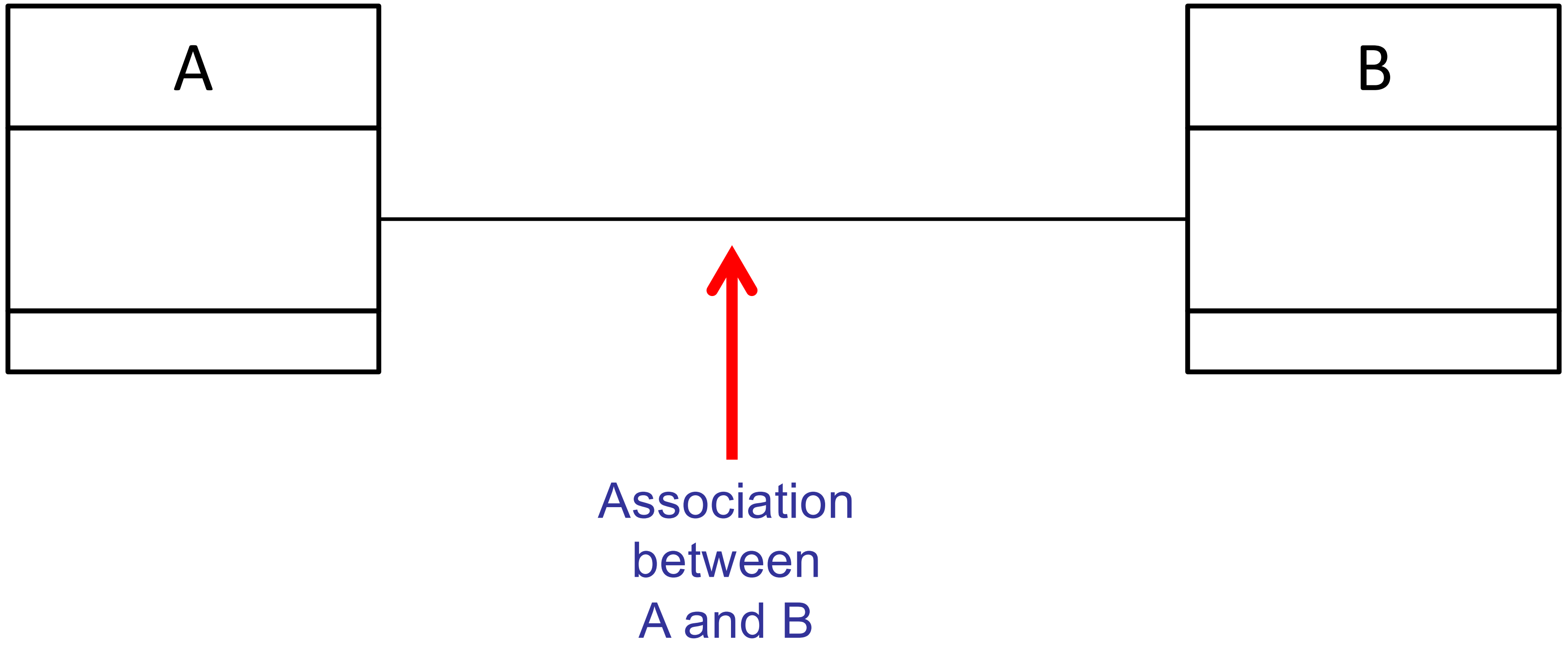
telephones: {0..*}: char

4.2. Conceptual Design

2.3. Association

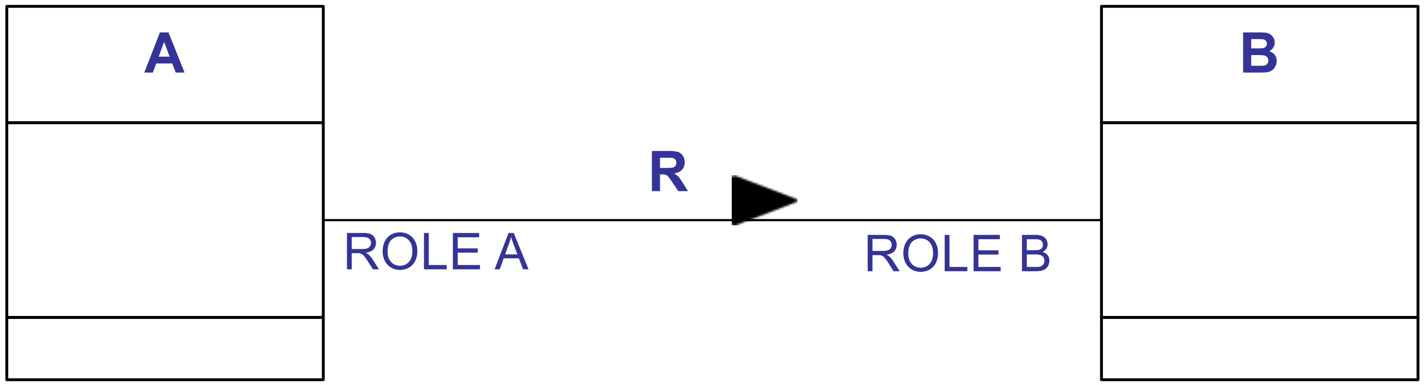
The associations represent relationships between the classes.

An association connecting two (and only two) classes is said to be **binary**.



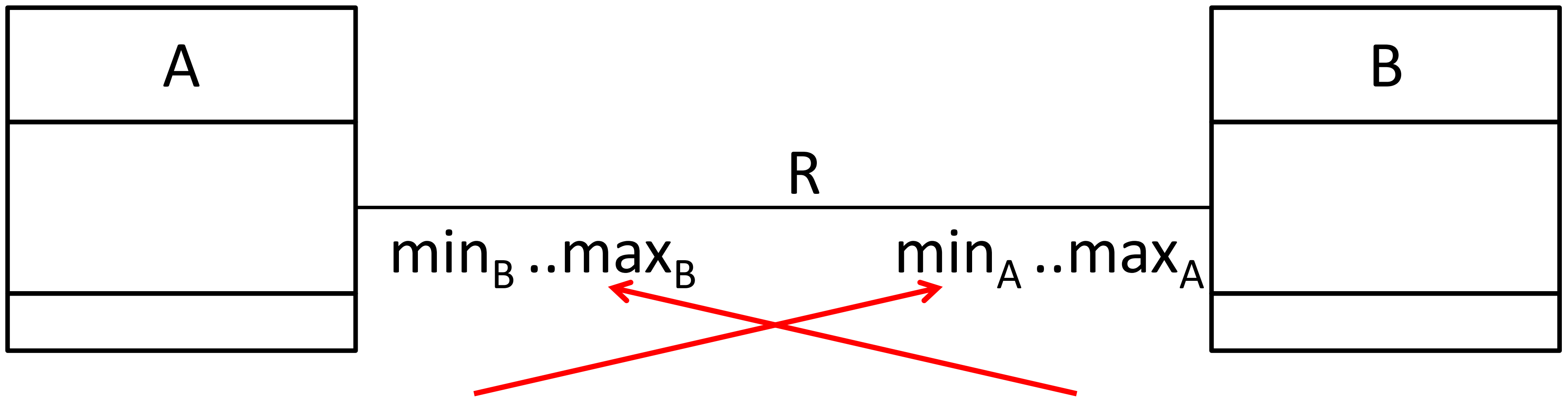
Associations are annotated with:

- **Name**: verbs are preferable
- **Arrow** (optional): to determine the subject and the object
- **Role** (optional): to clarify the relation, especially when the association is reflexive (a class is associated with itself)
- **Multiplicity**: participation-cardinality constraint.



Association multiplicity

***R*(*A*(min$_A$..max$_A$), *B*(min$_B$..max$_B$))**


A
B
R
$\text{min}_B .. \text{max}_B$
$\text{min}_A .. \text{max}_A$

Each occurrence of *A* is related to, at least, *min$_A$* occurrences of *B* and with, at most, *max$_A$* occurrences of *B*

Each occurrence of *B* is related to, at least, *min$_B$* occurrences of *A* and, at most, *max$_B$* occurrences of *A*

Multiplicity: Most usual values

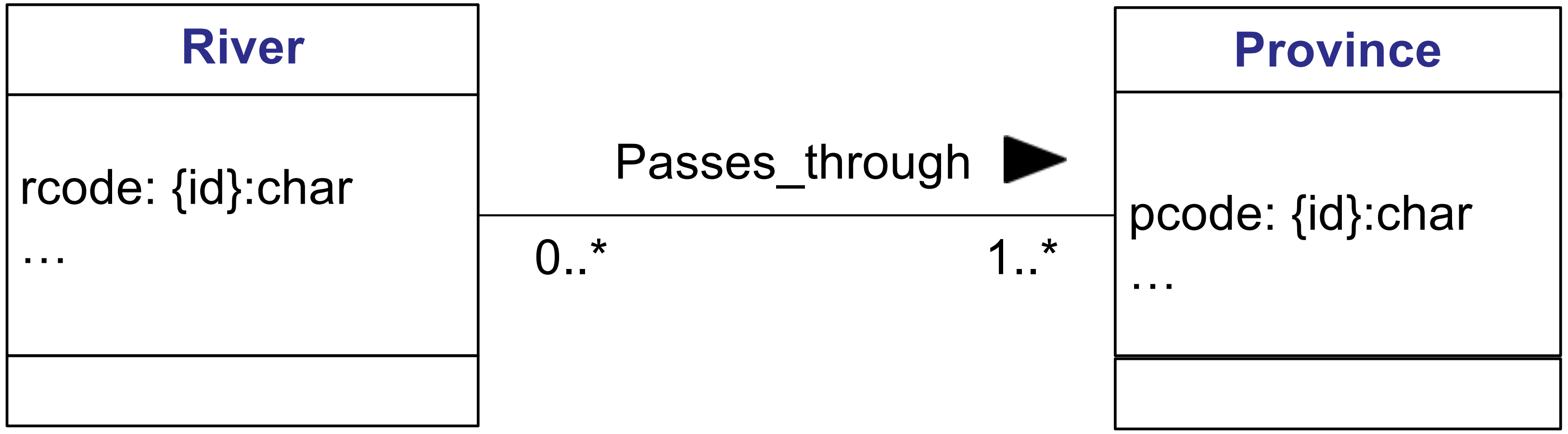
- min_A=1 y max_A=1
- min_A=0 y max_A=*
- min_A=0 y max_A=1
- min_A=1 y max_A=*

When the minimum multiplicity (min_A) of a class A is greater than 0 we say that class A has an **existence constraint** relative to the association.

Example

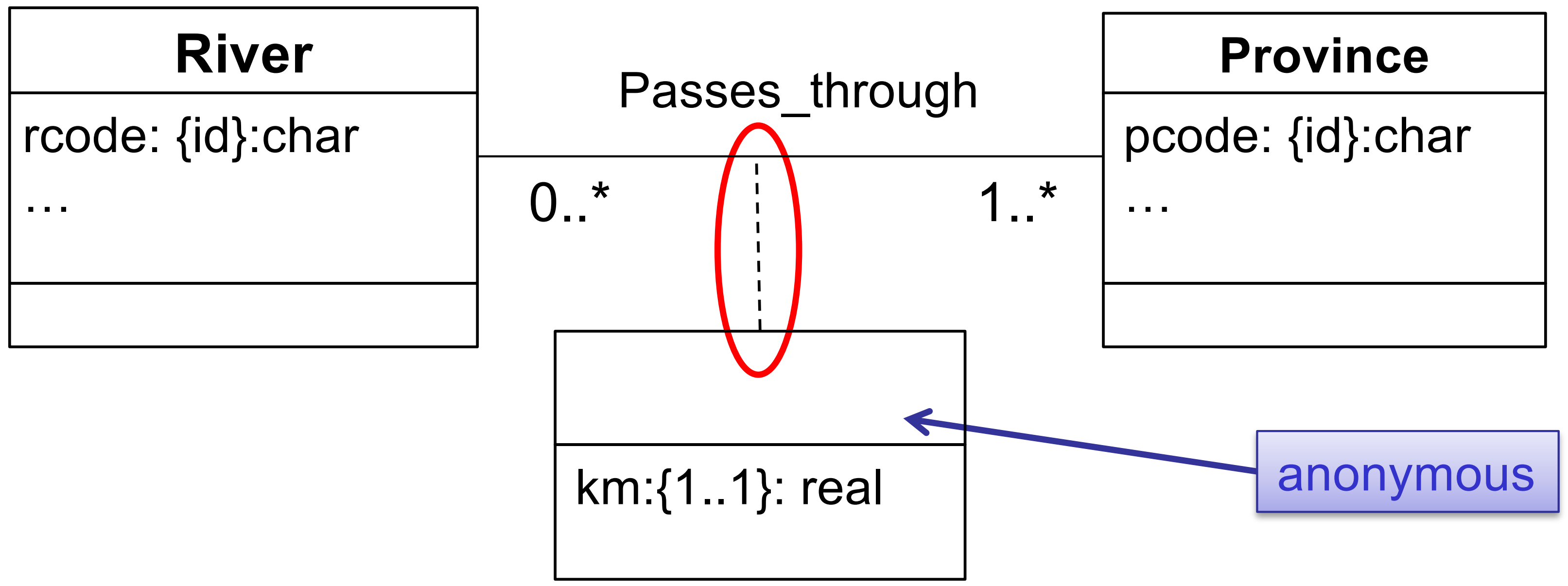
Diagram representing information about rivers passing through provinces:

“A river can pass through several provinces, at least one, and that a province can be crossed by several river or none.”



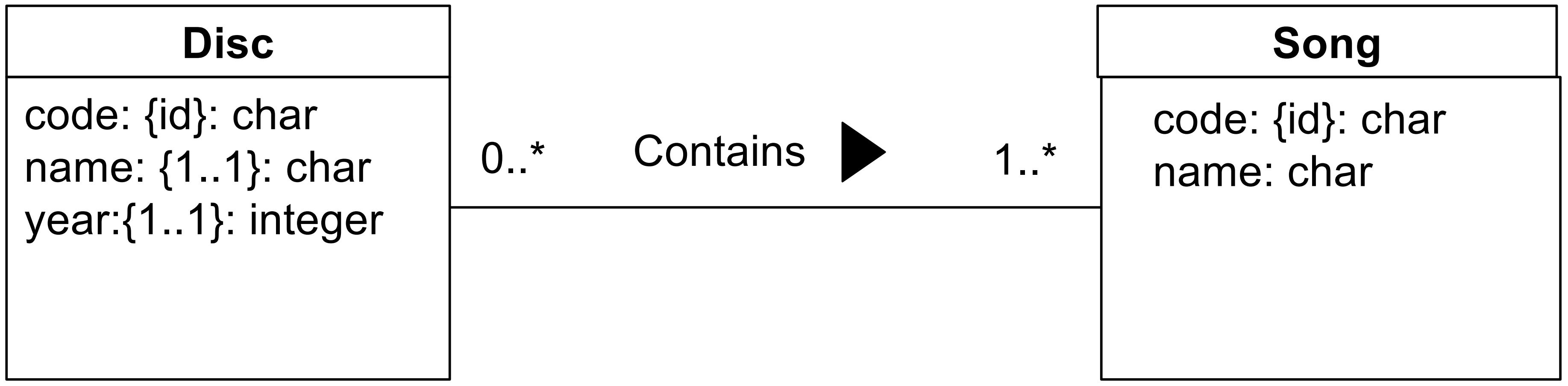
Anonymous class (link attributes)

We want to include the information: “number of kilometers the river goes though the province”.
The *Passes_though* association has a anonymous class or link attribute (dashed line) to the association.



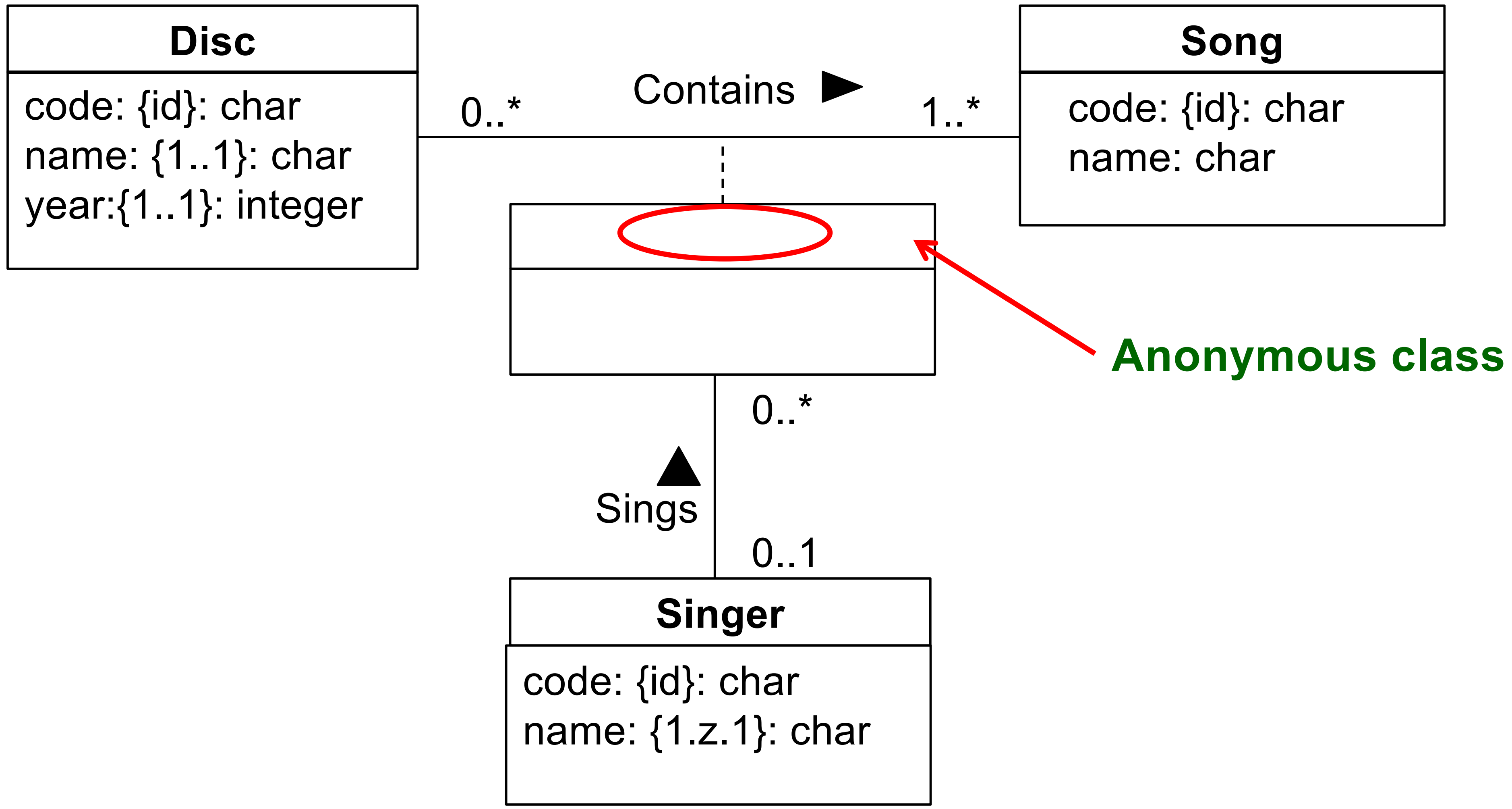
Association with associations

Consider the information about *discs* and *songs*, where the association indicates that a disc contains a song

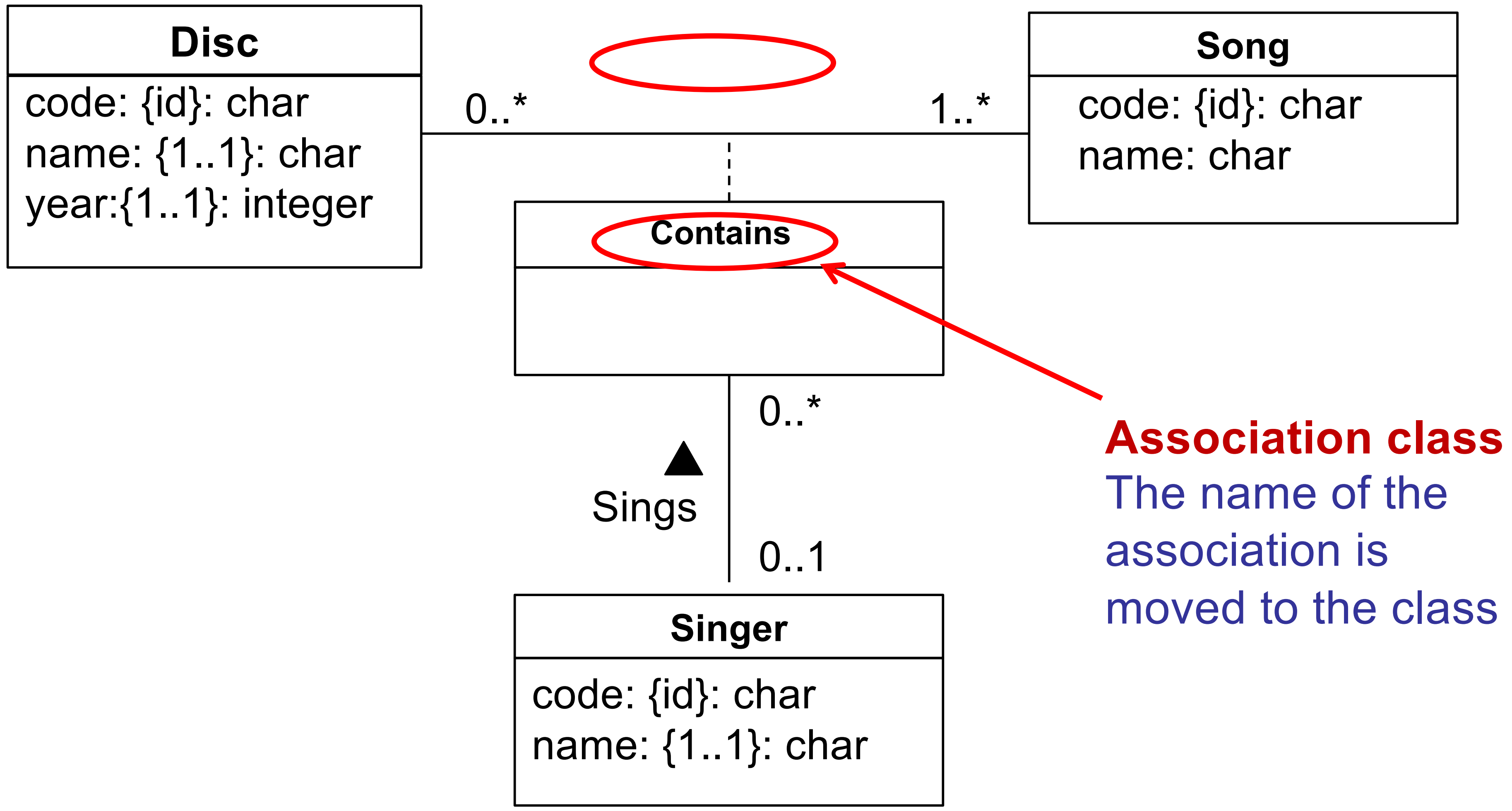


What if we also want information about singers and which song is sung by each singer in a given disc?

Solution 1: We use an association to the anonymous (empty) class



Solution 2: We use an association class



Association class
The name of the association is moved to the class

4.2. Conceptual Design

2.4. Weak classes

A class has a **identification dependency constraint** when it cannot be identified with its own attributes.

Its occurrences must be distinguished through the association to other class(es).

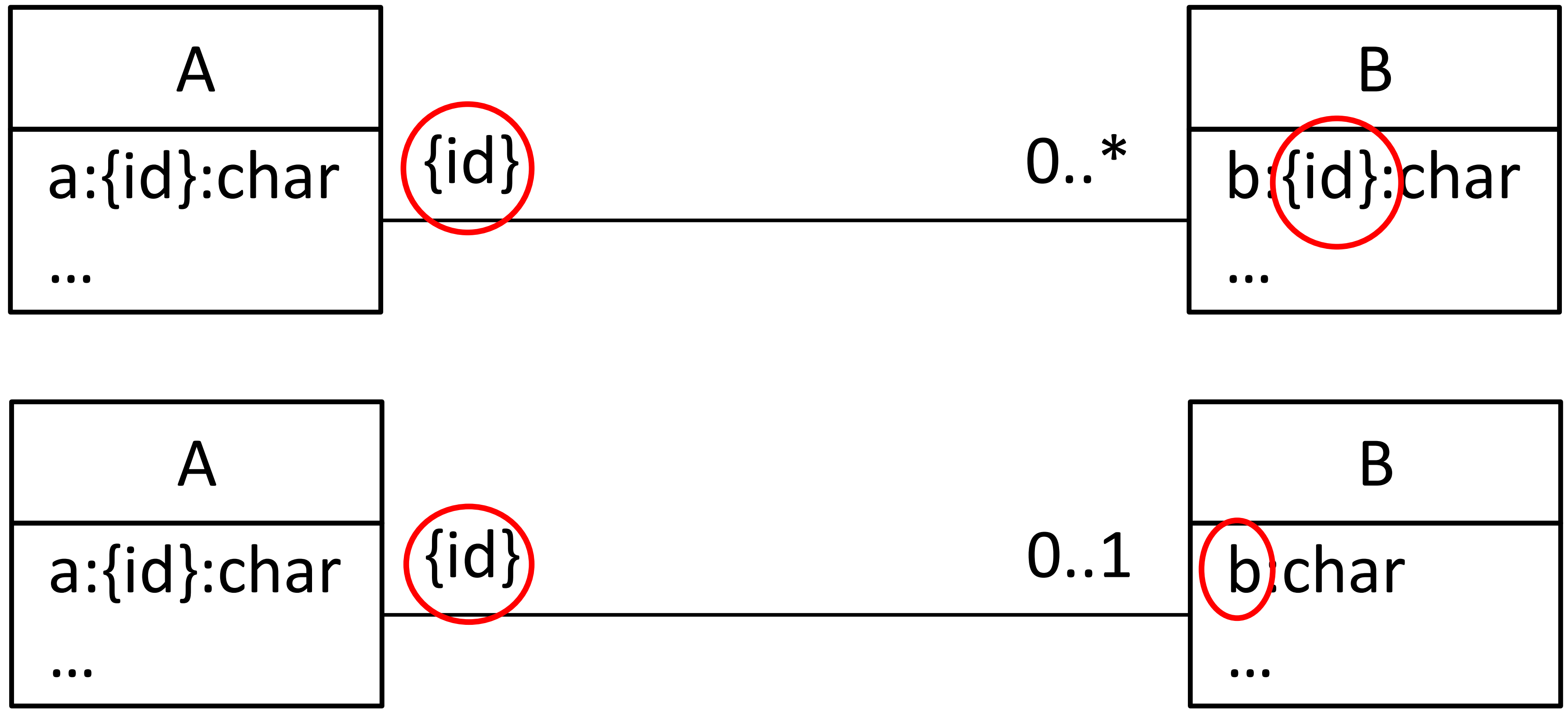
These classes are called **weak classes.**

When a weak class depends on another class (which might be weak as well), we talk about the **subordinate** and the **dominant** classes.

This constraint is represented by the label {id} instead of the multiplicity (“id” is a special case of ”1..1”):

The weak class counts on some attribute of the dominant class in order to be identified.

Diagrammatic representation



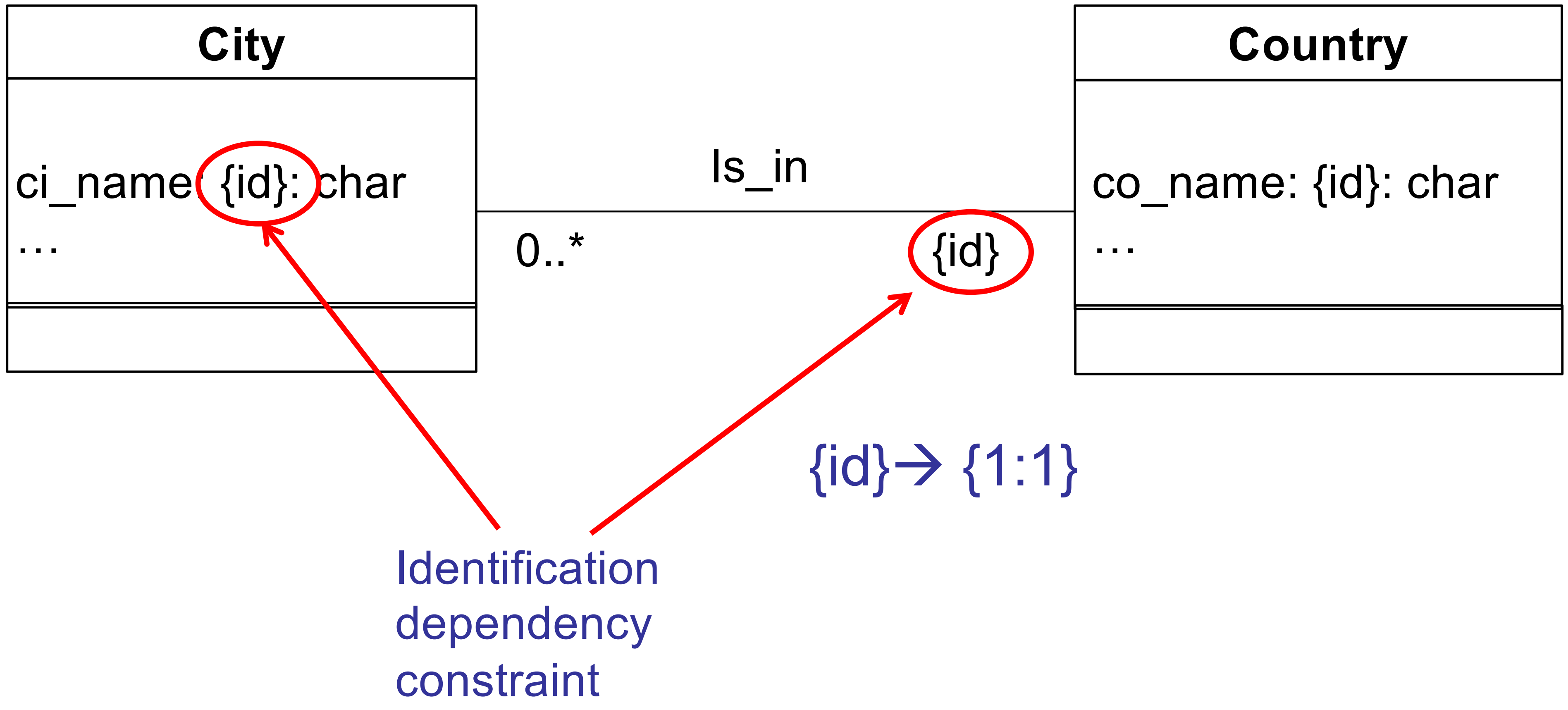
Strong (dominant) classes

Weak (subordinate) classes

Example

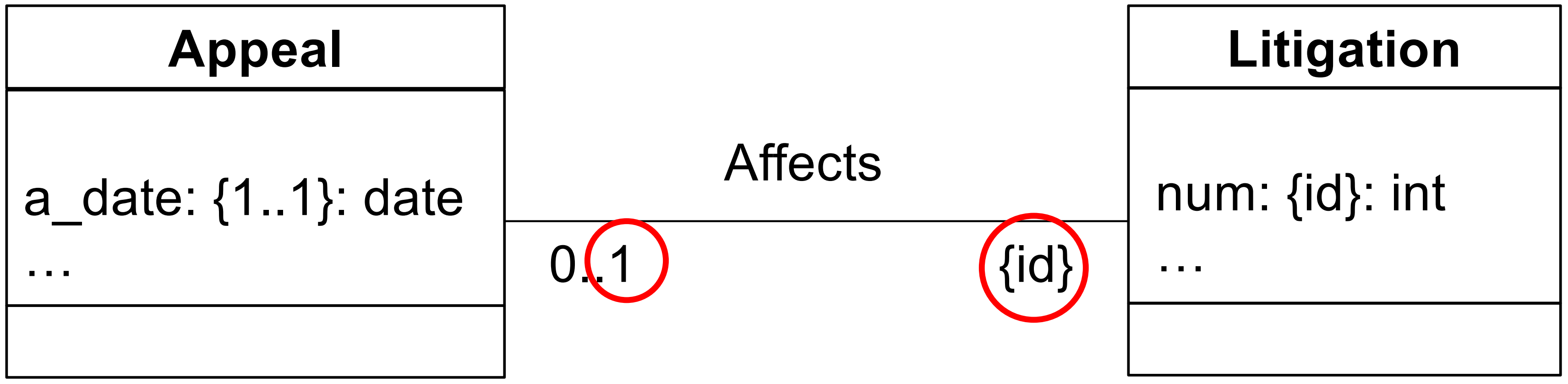
We need the *ci_nom* and the *co_name* to identify the city.

Valencia… In Spain or Venezuela?



The *ci_name* cannot identify the city on its own

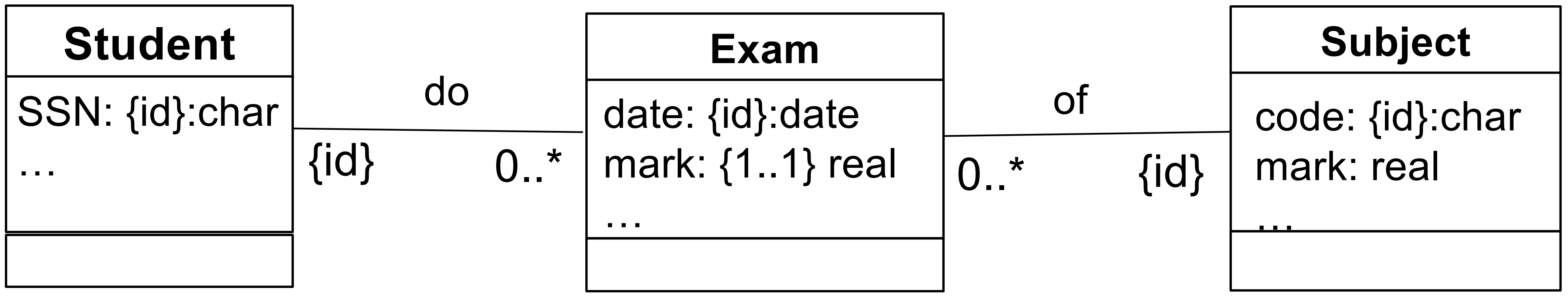
Example



Since a *litigation* can only have one *appeal*, we can just identify the *appeal* by the litigation number

Example

We want to organize the grades obtained by the students in several subjects. One student can do several exams for the same subject (but in different dates) and the exams have no identifier.



4.2. Conceptual Design

2.5 Ternary associations

An association connecting two classes is called **binary**.

In UML, we can also represent N-ary associations Nonetheless, for simplicity, we are going to use only binary associations.

Ternary associations are usually **represented** using:

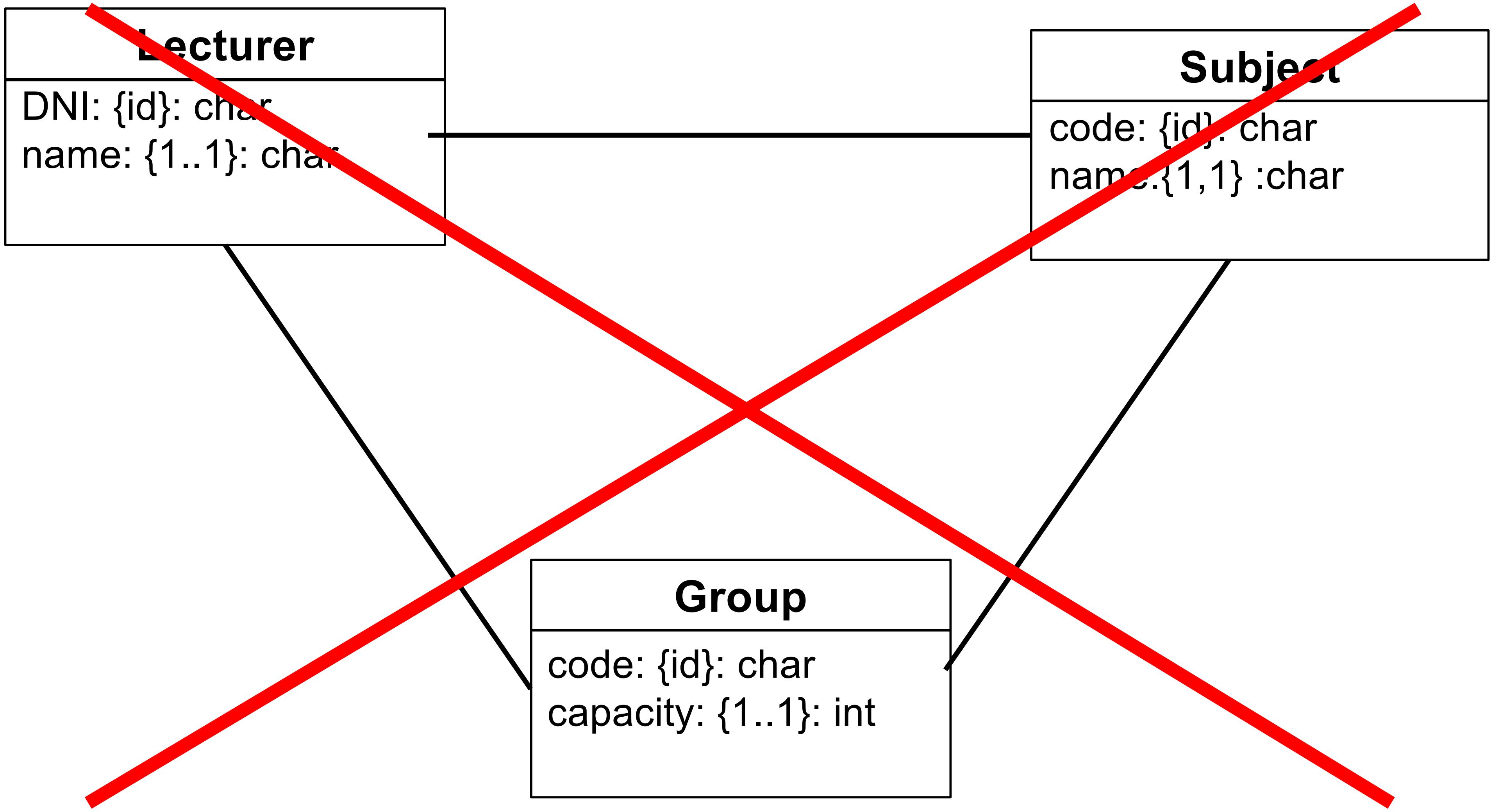
- association classes
- weak classes

Example:

A ternary association between lecturer-subject-group:

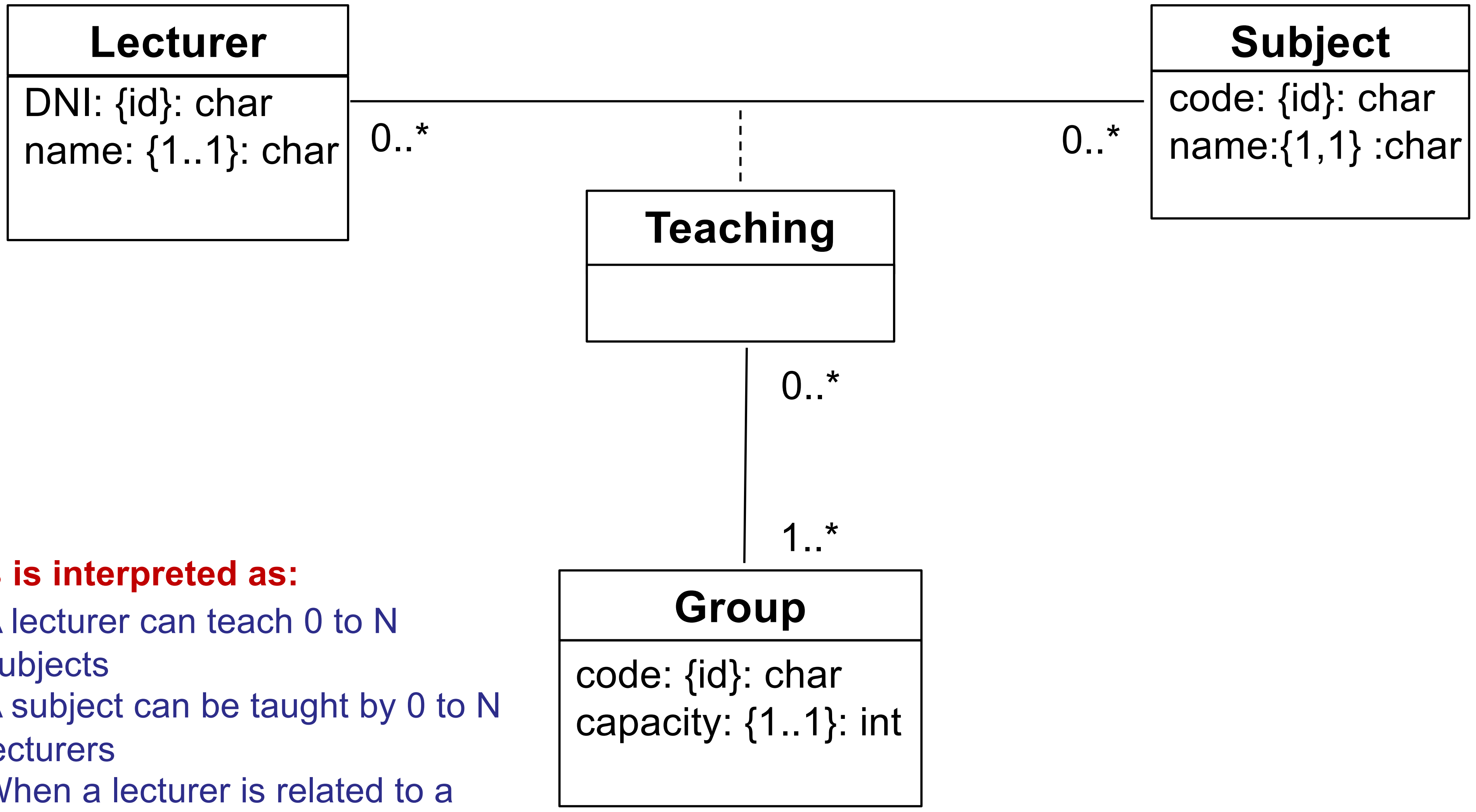
- A lecturer can teach any number of subjects in any number of groups
- A subject can be taught by any number of lecturers and in any number of groups
- A group can have any number of subjects taught by any number of lecturers

It cannot be modelled using 3 binary associations:



A lecturer teach several subjects. A lecturer teach several groups but,
Which subject is taught in each group by a teacher ?

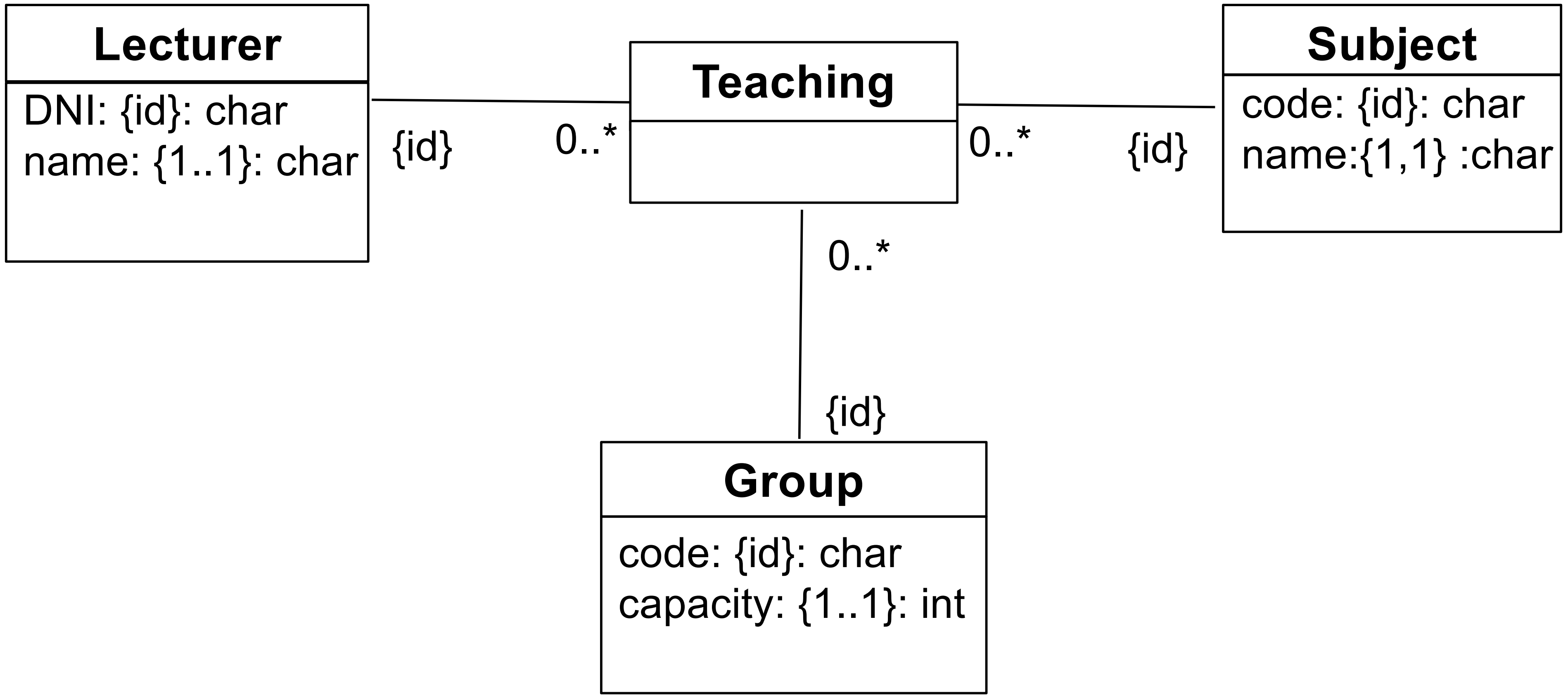
A) Ternary association with **association class**



This is interpreted as:

- A lecturer can teach 0 to N subjects
- A subject can be taught by 0 to N lecturers
- When a lecturer is related to a subject, it must have at least one associated group.
- A group is related to 0 to N pairs of lecturers and subjects.

B) Ternary association with a **weak class**



4.2. Conceptual Design

2.6 Specialization / Generalization

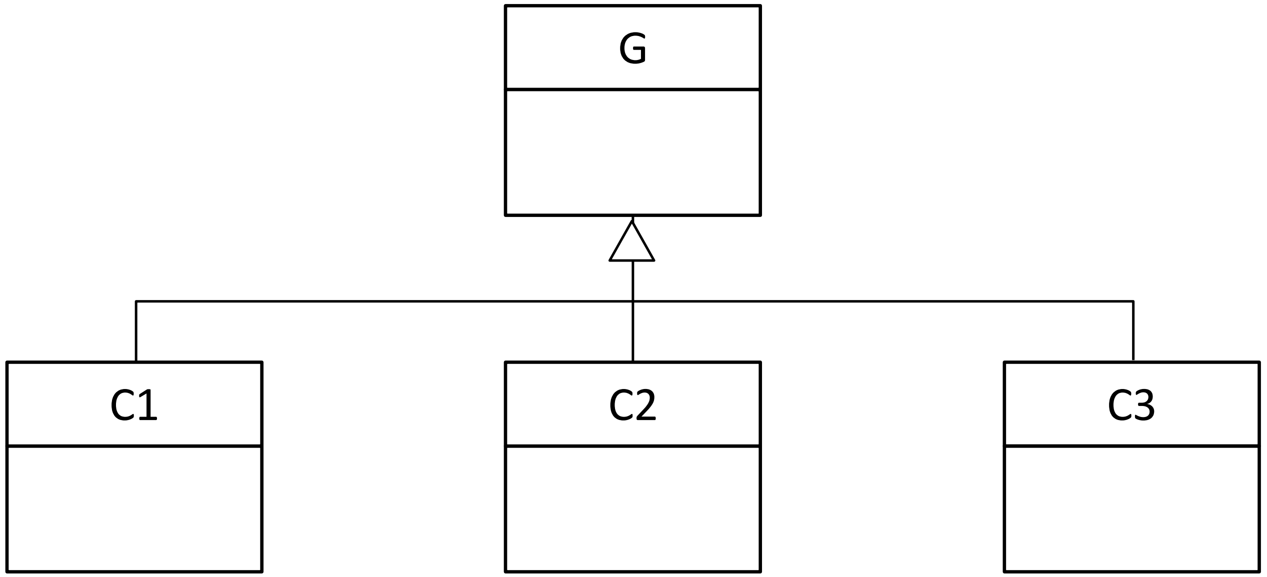
When several classes share properties that might well be represented as part of a more abstract class, taking the common properties, then we have a **generalisation/specialization association**.

- The general class is said to be specialized into one or more subclasses, and the specific classes are said to be generalized into a superclass.

- All subclasses ***inherit*** the attributes of the superclass.

* In UML, one subclass can repeat (and hence redefine) an inherited attribute. However, in database modelling, we are not considering this possibility.

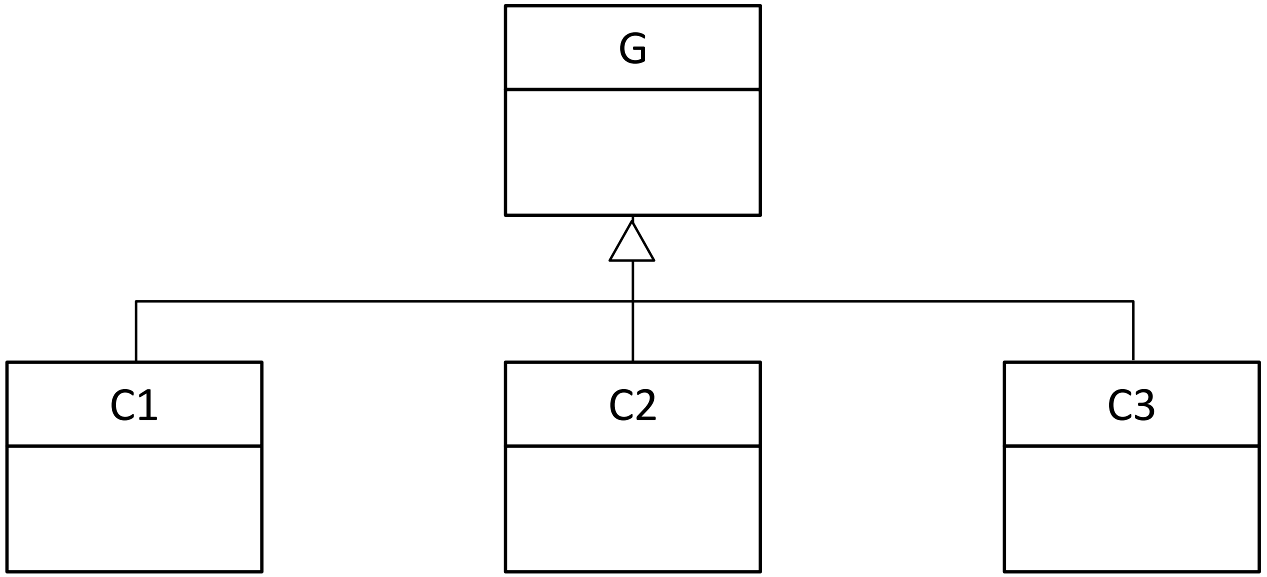
Kinds of generalization/specialization



Total (complete): Every occurrence of the superclass G must participate as an occurrence of any of its subclasses *C*1, *C*2 o *C*3

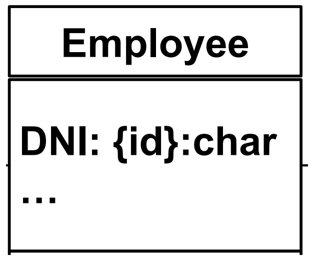
Partial (incomplete): There can be occurrences of the superclass G which are **not** of any of its **subclasses** (default case)

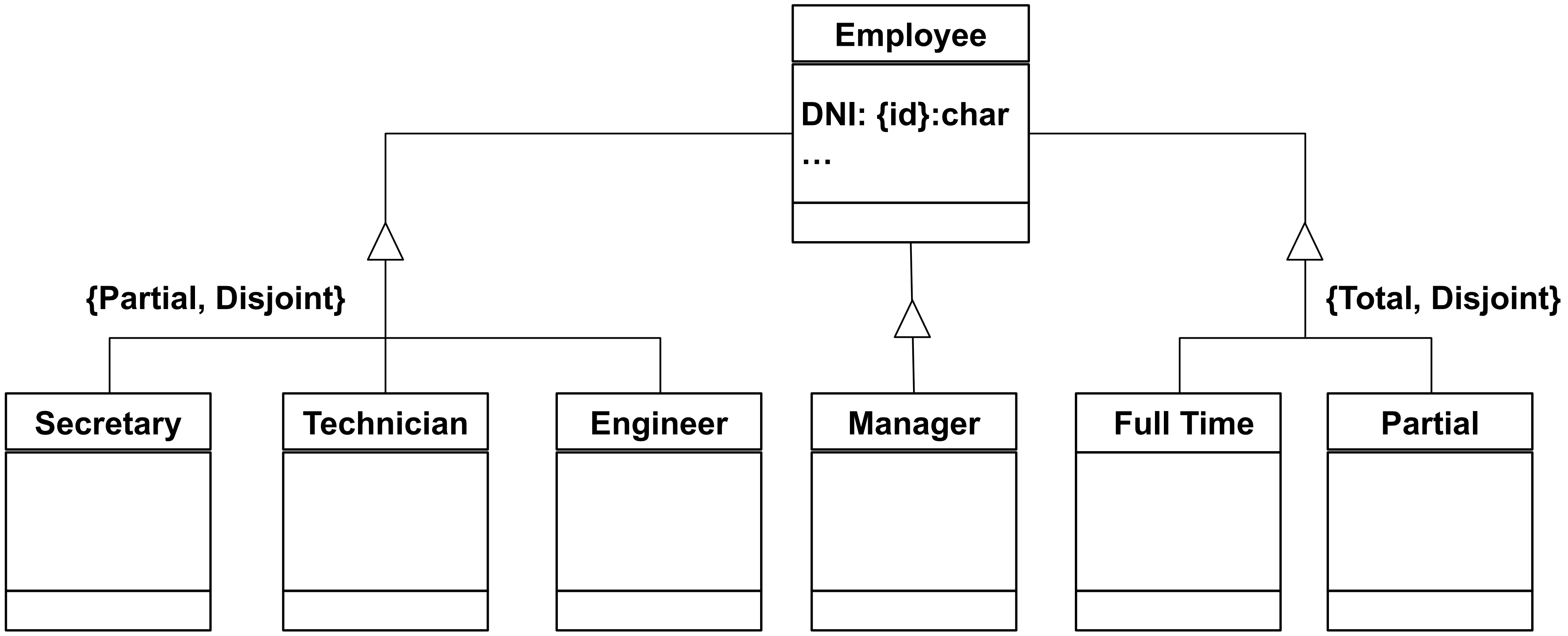
Kinds of generalization/specialization



Disjoint: One occurrence of the superclass G cannot participate as more than one occurrence of any subclasses.

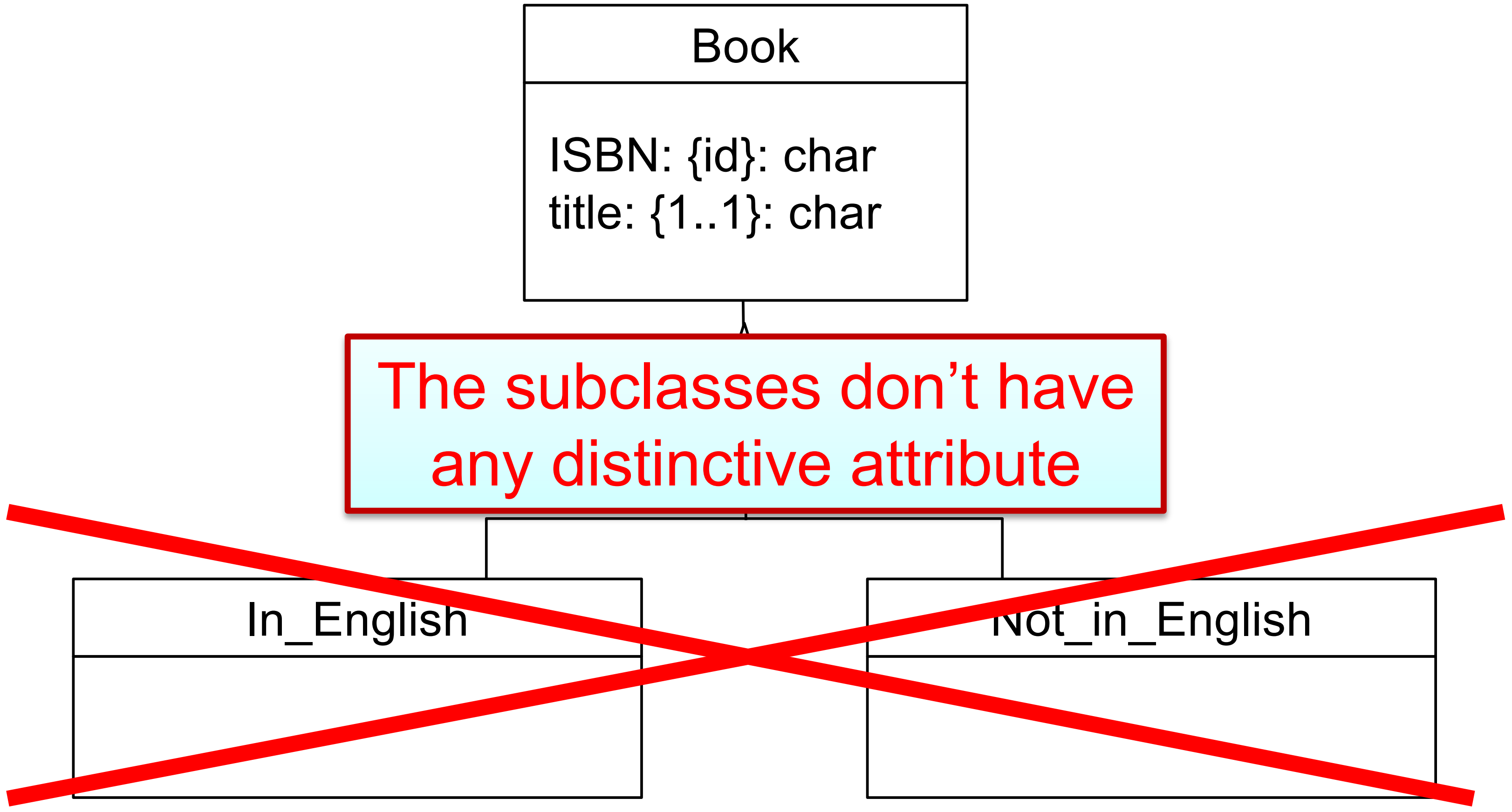
Overlapping (nondisjoint): One occurrence of the superclass G can participate as more than one occurrence of any of its subclasses (default case)


Employee
DNI: {id}:char
…


Employee
DNI: {id}:char
…
{Partial, Disjoint}
{Total, Disjoint}
Secretary
Technician
Engineer
Manager
Full Time
Partial

Don’t abuse specialization

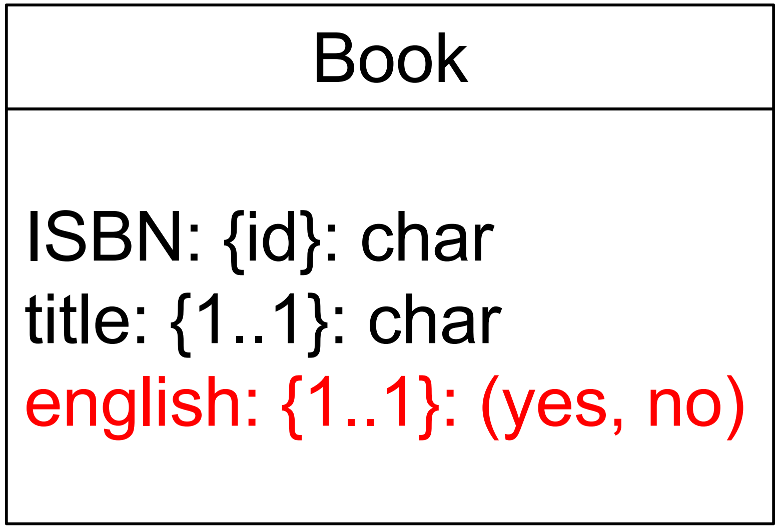
Books are classified in a library using their ISBN and tittle.
Books can be written in English or in other language.



The subclasses don’t have any distinctive attribute

Don’t abuse specialization

Books are classified in a library using their ISBN and tittle.
Books can be written in English or in other language.



4.2. Conceptual Design

3. Methodology to obtain the ER-UML diagram

1. Identify classes with their attributes
2. Identify generalization/specializations
3. Identify associations between classes
4. Specify integrity constraints
5. Final checks

Note that there will be choices leading to different (but valid) variations produced by different designers for the same problem

1. Identify classes with their attributes

For each object or entity of interest found in the requirements, we will define a class in the UML diagram

We will identify the attributes which define each class.

For each attribute we must:

- Associate a domain or, it is a derived attribute, a formula/expression of how it is defined.
- Specify the multiplicity and the uniqueness constraints

- Chose a class identifier. If there is no proper identifier, the class is considered weak and we must determine which dominant class/es it relies on to identify its occurrences.
- Check the diagram and reconsider the whole picture (redefine some class if it is necessary)

2. Identify generalizations/specializations

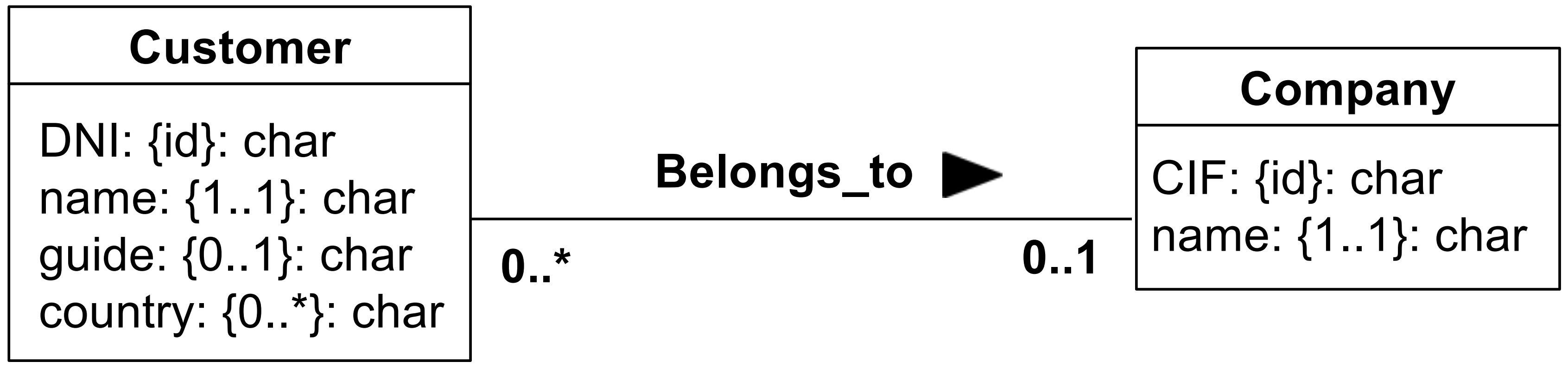
A. **Descending strategy**: Specialization of a general class

B. **Ascending strategy**: Generalization of several specific classes

C. **Hierarchization (is_a)**: One class is a special case of the other

A. Descending strategy: Specialization of a general class

Travel agency

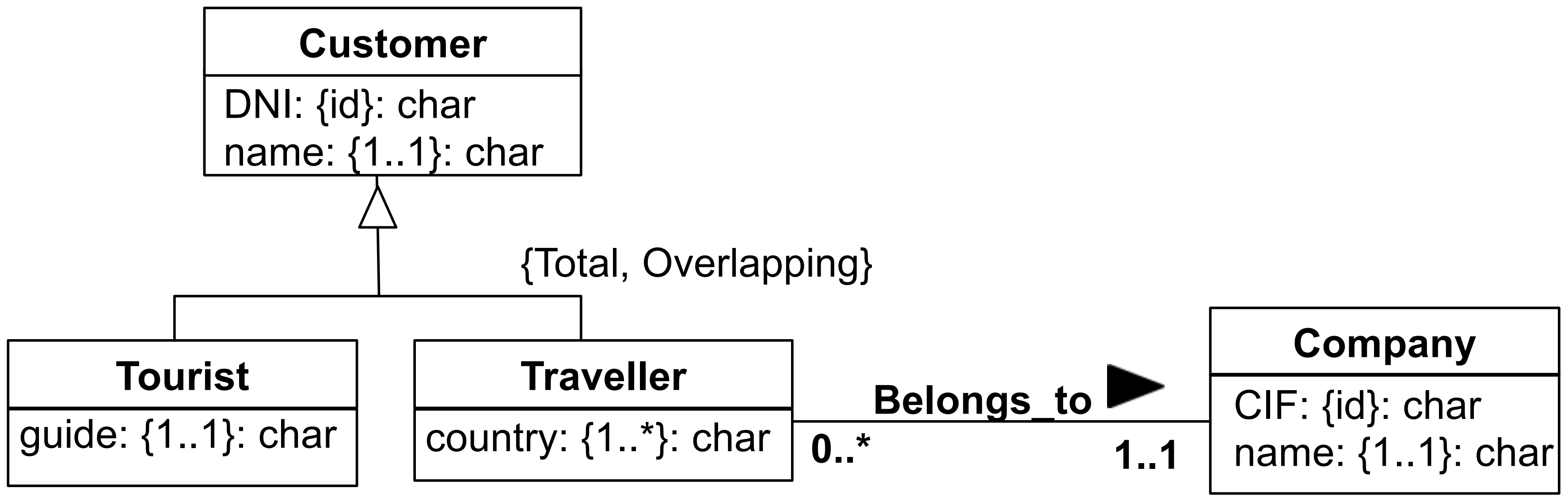


We are said that there are two kinds of customers:

- tourists, who are always accompanied by a guide and do not come through a company; and
- business travellers, who always belong to a company and we need to know which countries they visit.

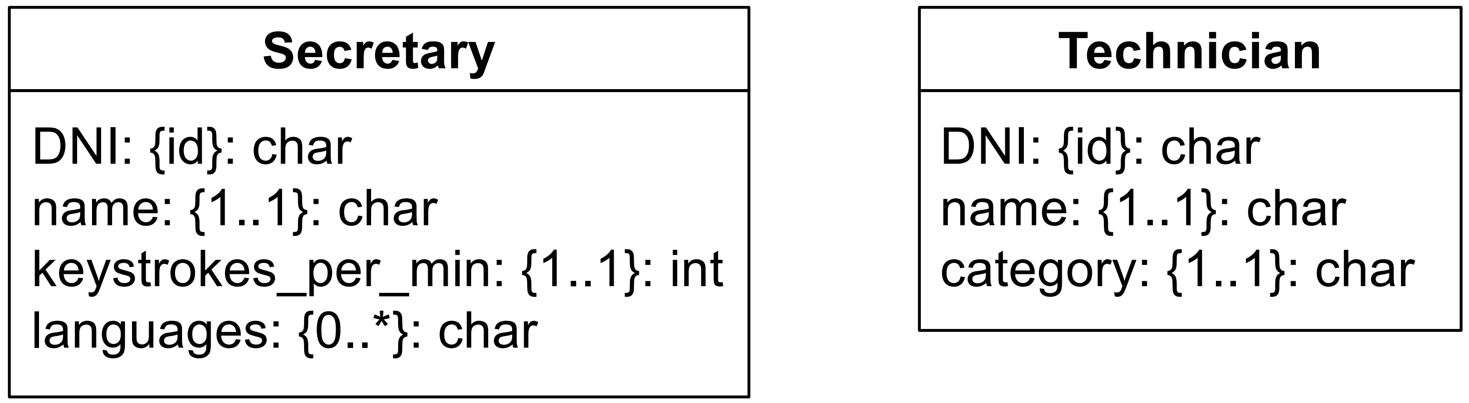
A. Descending strategy: Specialization of a general class

A better solution:



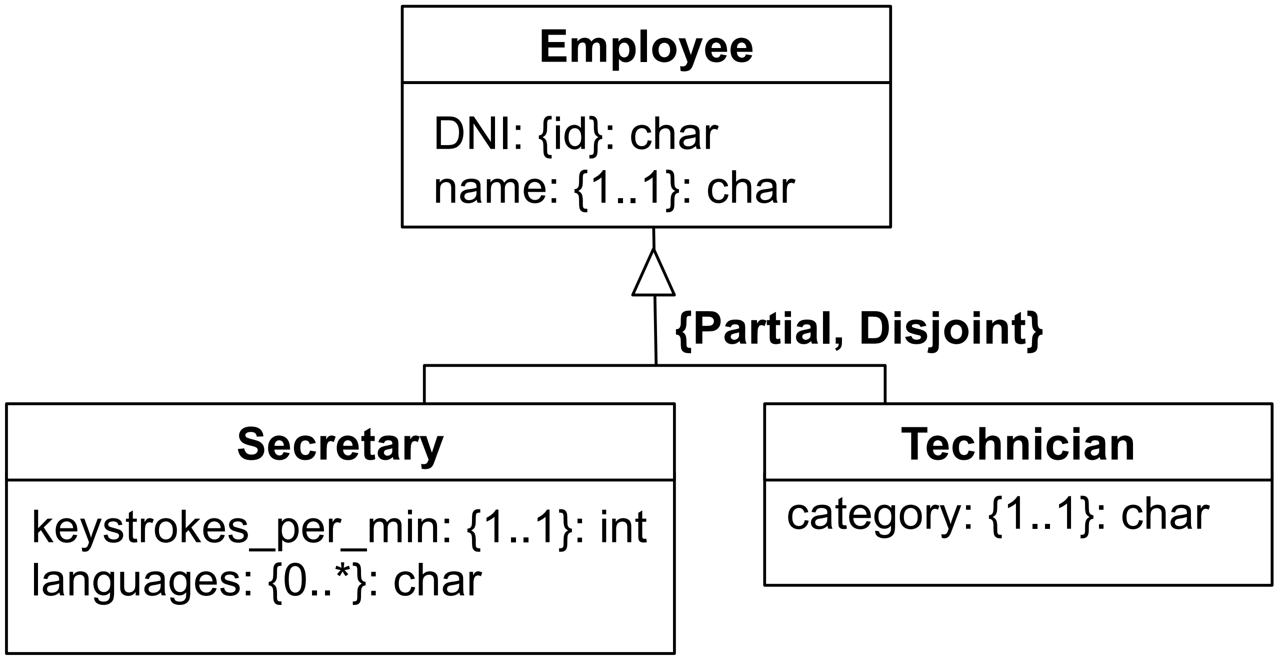
Now we have much more adjusted ranges for the multiplicities.
Unused attributes have been eliminated.

B. Ascending strategy: Generalization of several specific classes



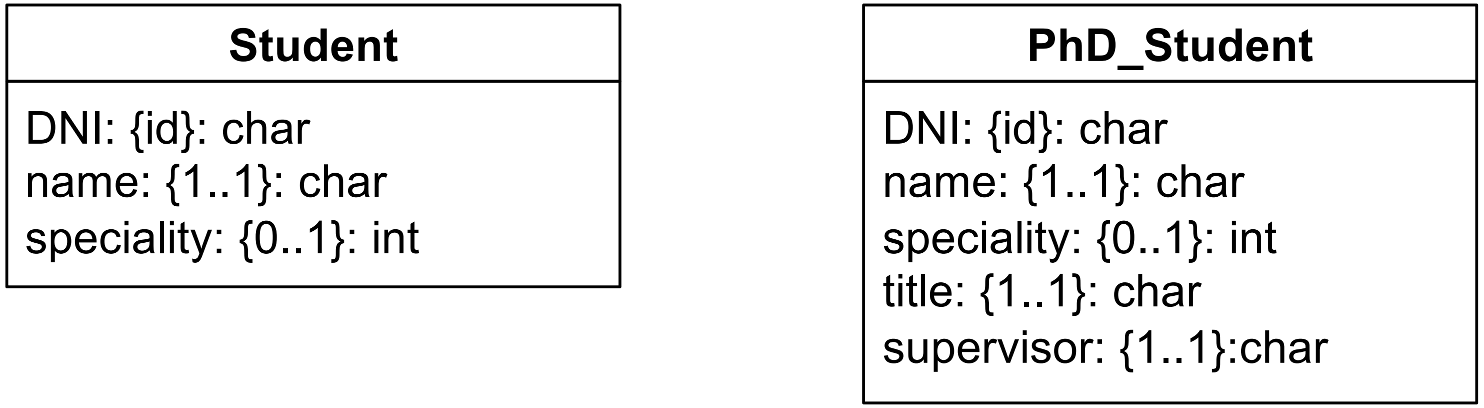
If we also observe that both classes refer to **employees** of a company for whom we may be interested to perform several processes together…

B. Ascending strategy: Generalization of several specific classes



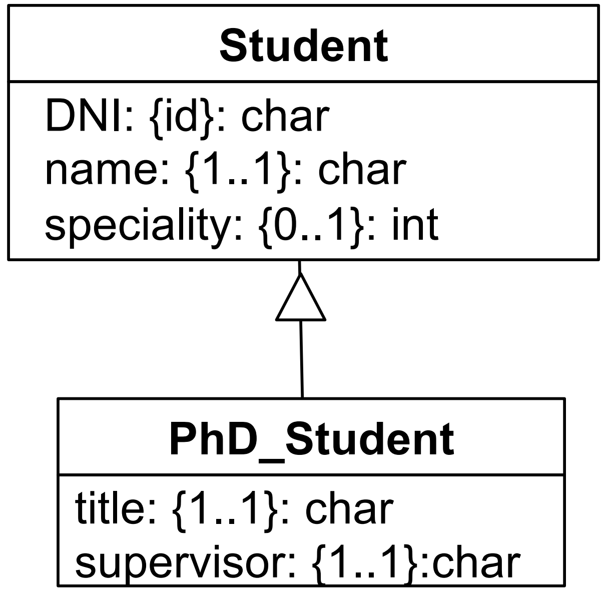
If we also observe that both classes refer to employees of a company for whom we may be interested to perform several processes together…

C. Hierarchization (is_a)



If we know that every PhD student is also a student, a better solution could be obtained

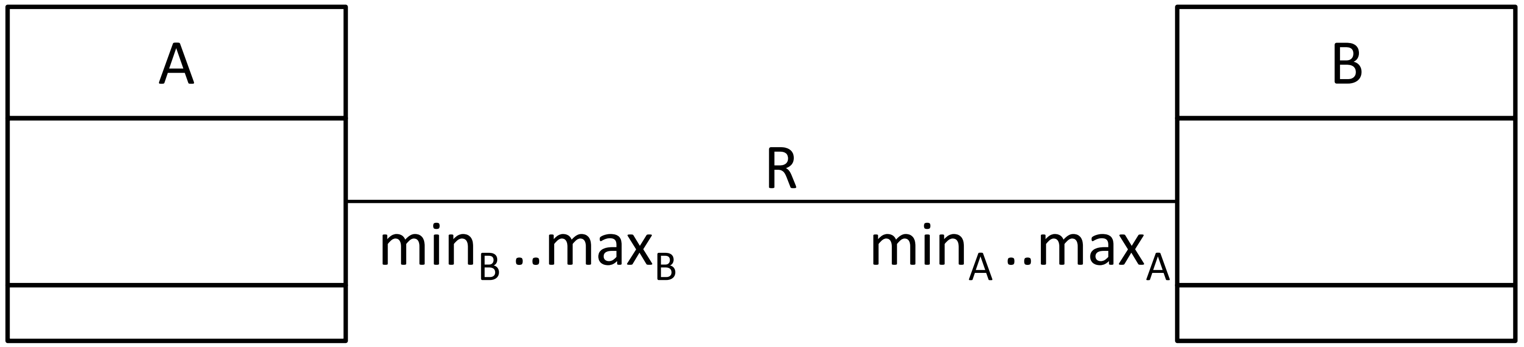
C. Hierarchization (is_a)



Which eliminates two specific attributes for regular students.

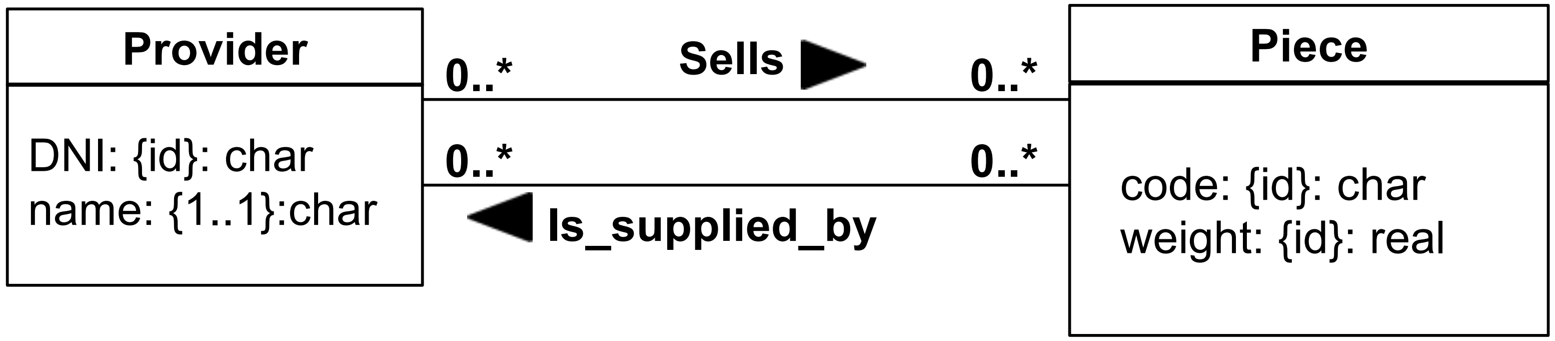
3. Identify associations between classes

3.1. Determine the **multiplicity** (minimum and maximum cardinality)



In case of doubt, choose the least restrictive multiplicities

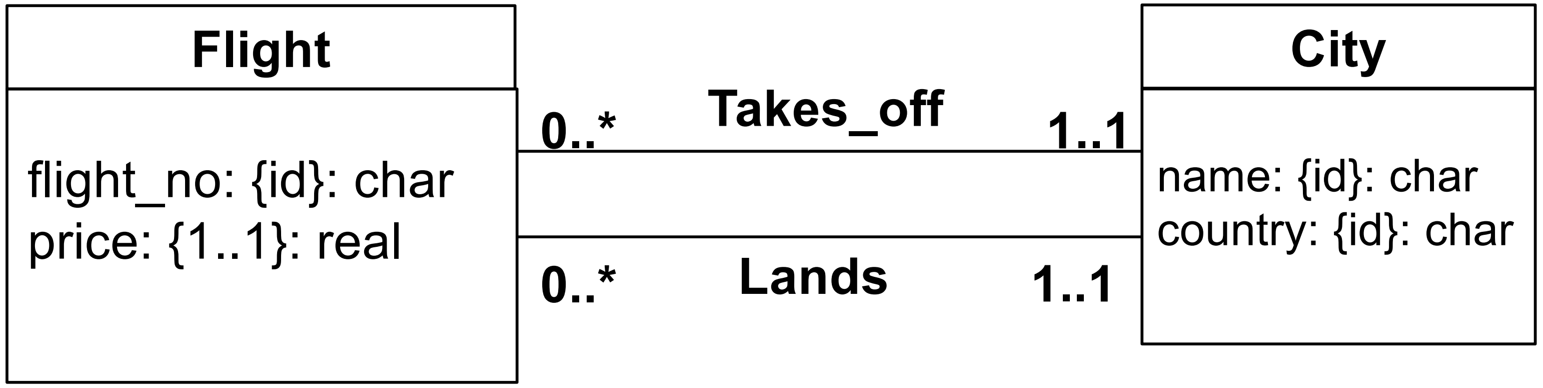
3.2 Remove redundant associations



Eliminate one
of the associations

There are two associations but, despite their different names, they represent the same information

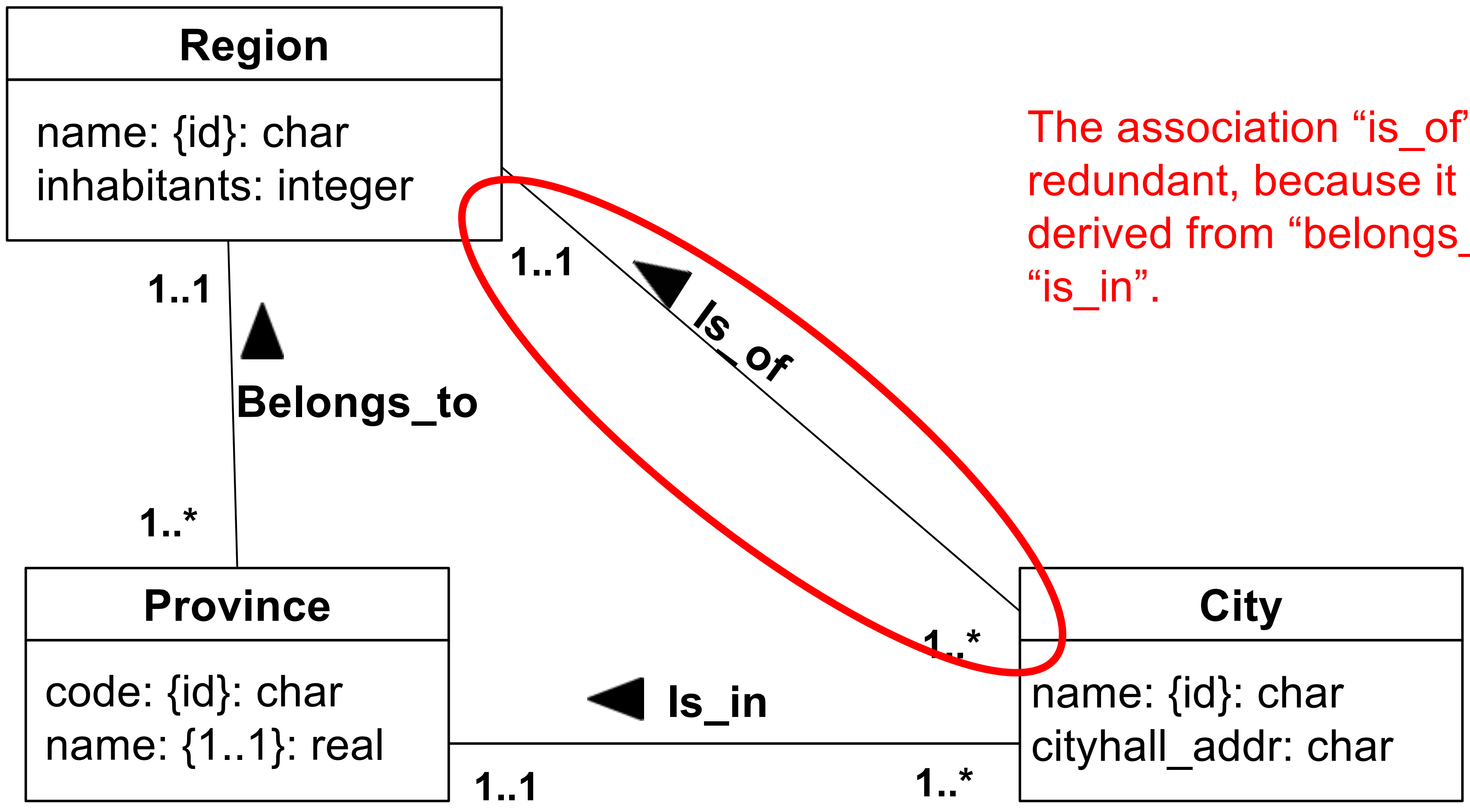
3.2 Remove redundant associations



In some occasions, double associations are perfectly meaningful because they represent different information

3.3 Remove transitive redundancies

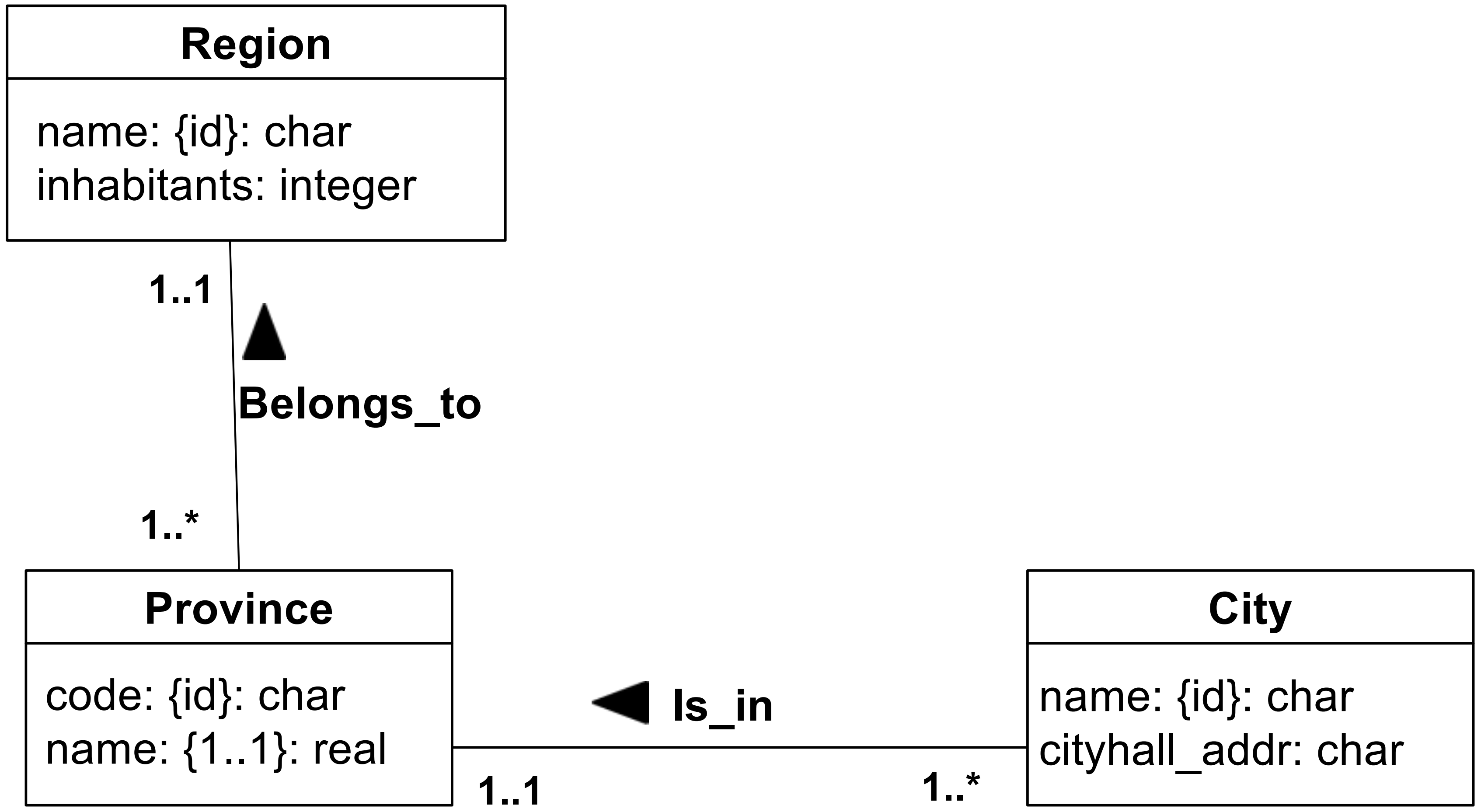
An association is redundant because it can be derived from others associations



The association “is_of” is redundant, because it can be derived from “belongs_to” and “is_in”.

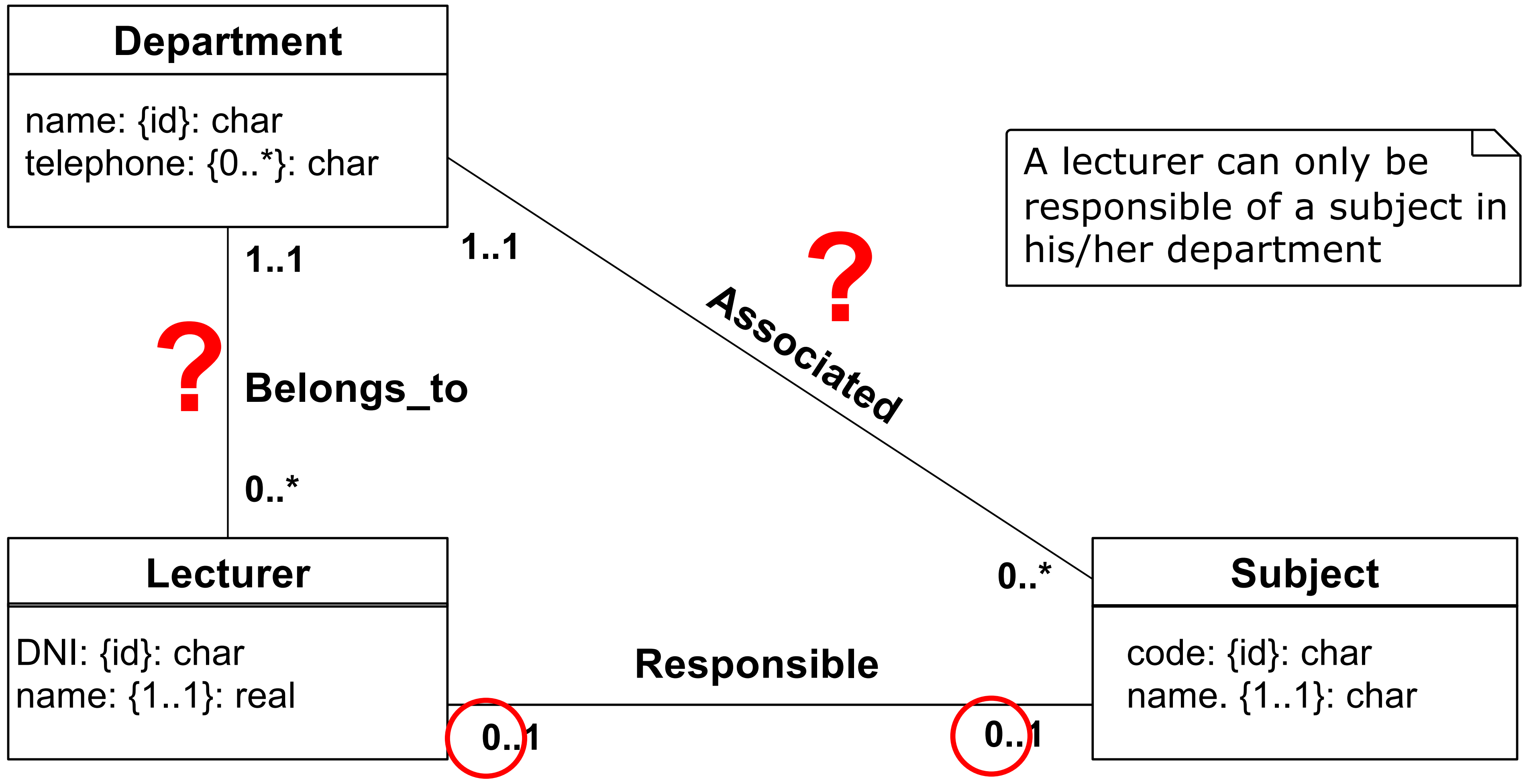
3.3 Remove transitive redundancies

An association is redundant because it can be derived from others associations

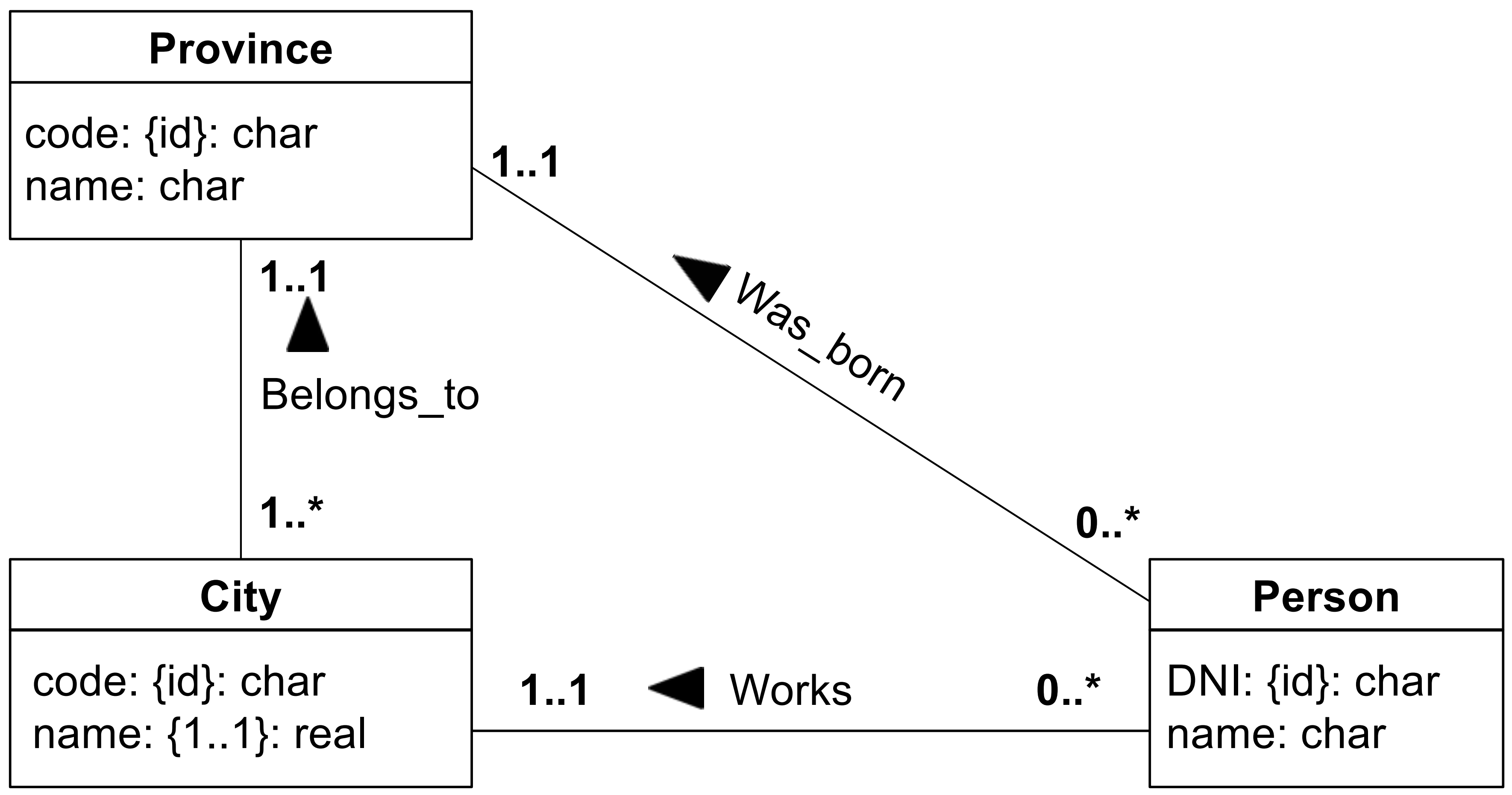


3.3 Remove transitive redundancies

Sometimes is not possible to eliminate any association

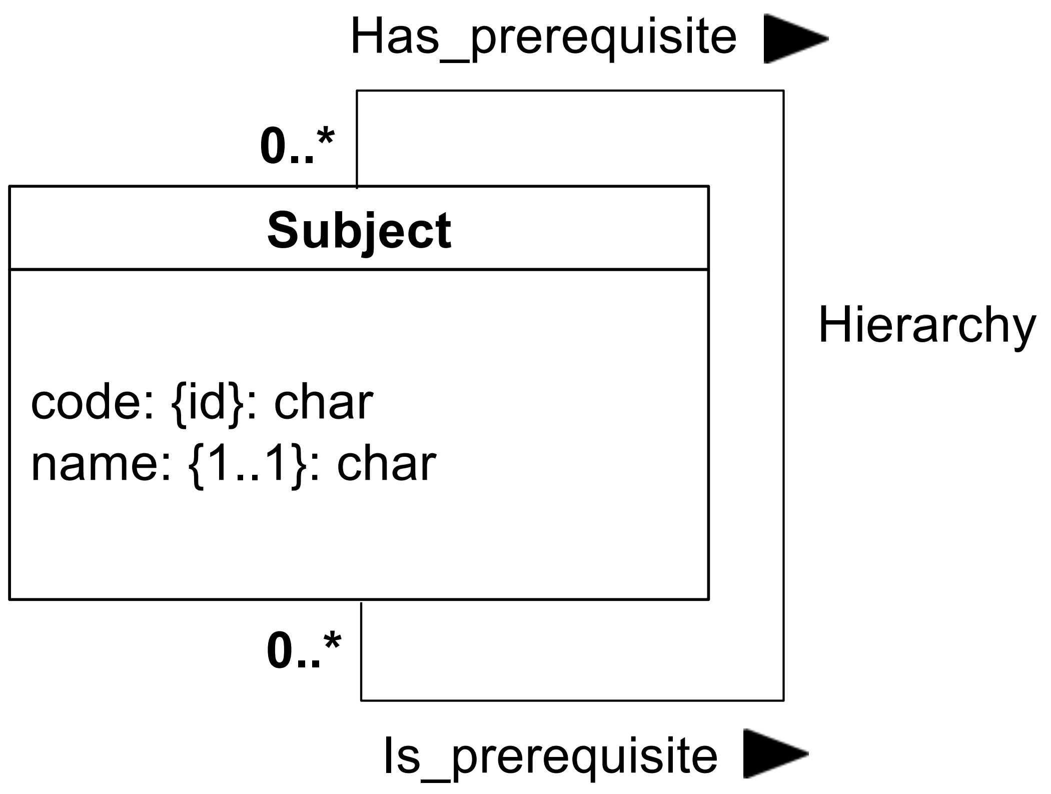


3.3 Remove transitive redundancies



Do not eliminate:
A person could have been born in a province and work in other:
There is no transitivity

3.4 Specify roles in associations of a class with itself (reflexive)



3.4 Specify roles in associations of a class with itself (reflexive)

Be aware of the reflexive associations:

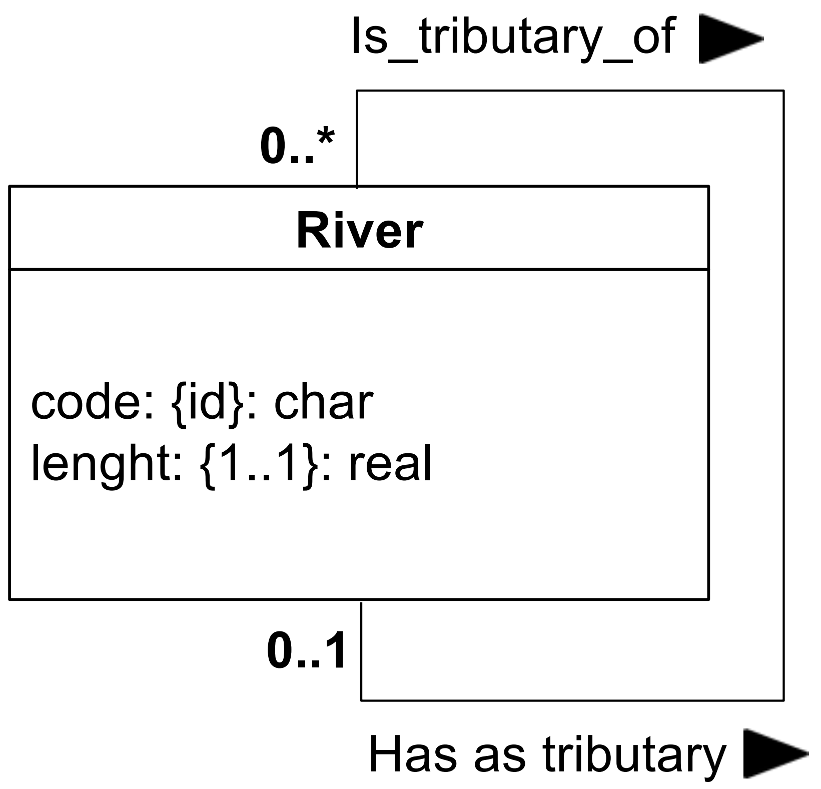
We usually have to indicate some properties which don't appear in the association definition

In the previous example (subject prerequisites)

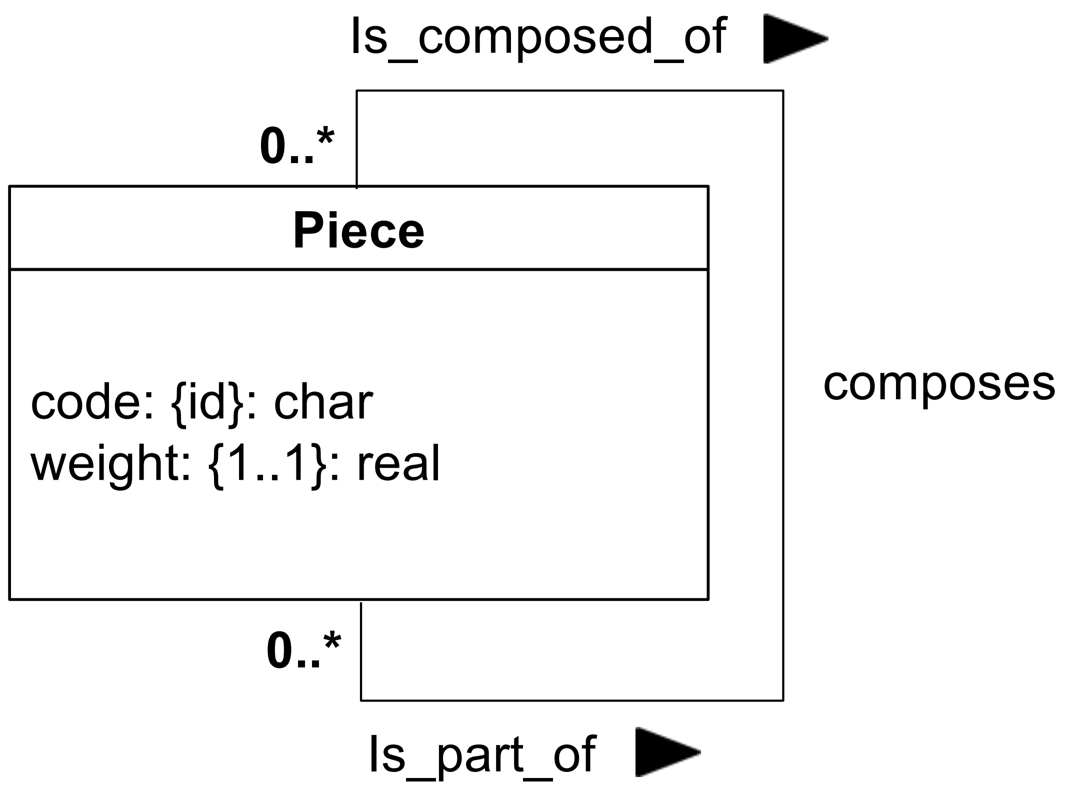
- It is antireflective:
  A subject cannot be prerequisite of itself.
- It is antisymmetric:
  A subject cannot be prerequisite of a prerequisite of itself

But these properties are **not represented in the UML** diagram.

3.4 Specify roles in associations of a class with itself (reflexive)



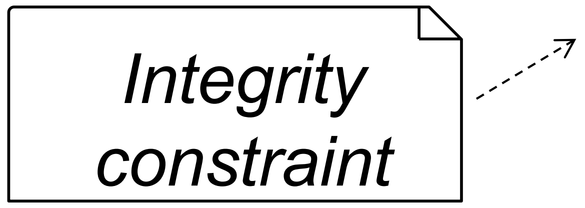
3.4 Specify roles in associations of a class with itself (reflexive)



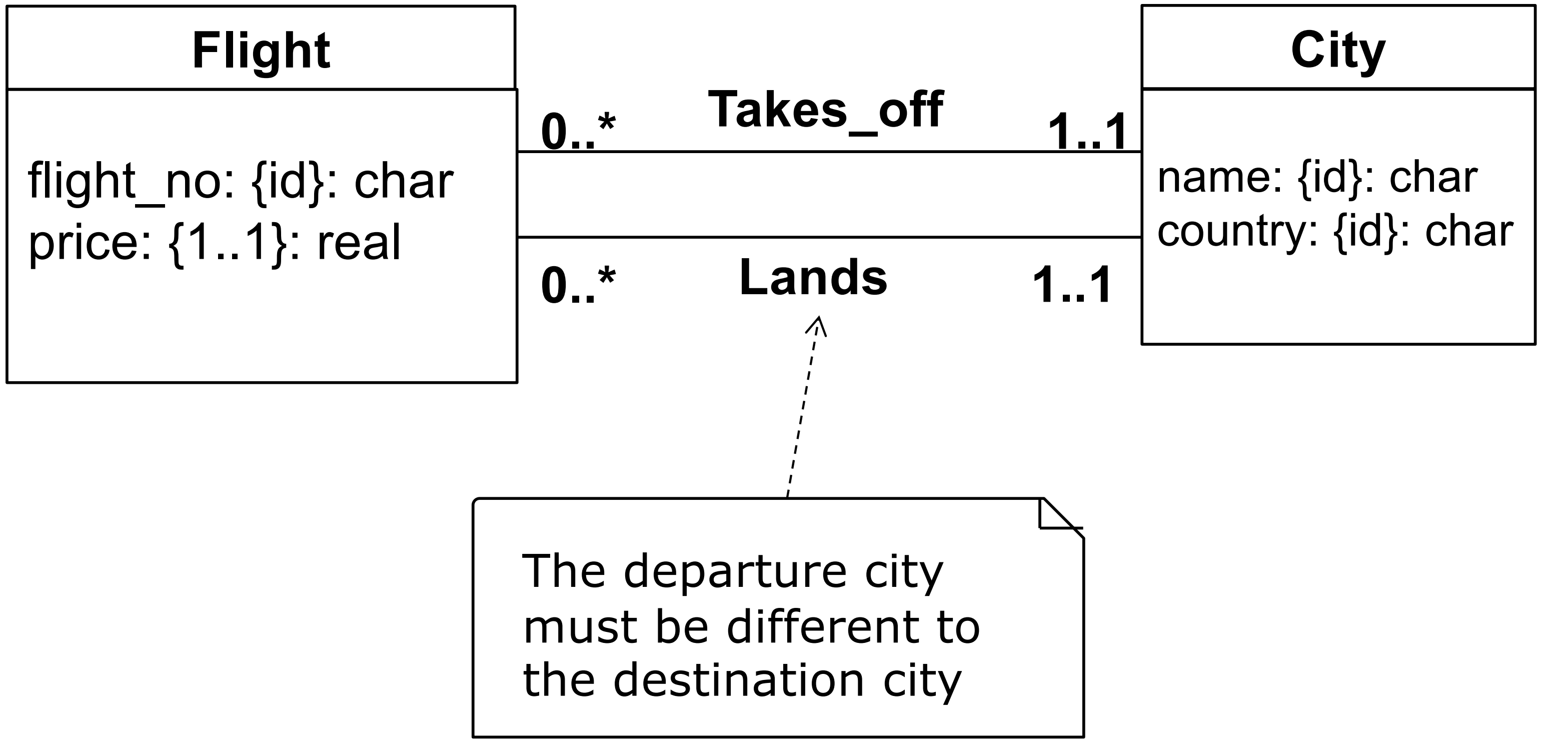
4. Specify integrity constrains

Any other property of reality that has not been expressed in the UML diagram must be included now.

Use UML **annotation elements**:



Annotations can also be used for other purposes: comments, clarifications, observations, etc.


Flight
flight_no: {id}: char
price: {1..1}: real
0..*
Takes_off
1..1
0..*
Lands
1..1
City
name: {id}: char
country: {id}: char
The departure city must be different to the destination city

5. Final checks

- Names:
  - We cannot use the **same** name for **different classes**, **associations** or **roles**.
  - The **attribute names** in a class cannot be **repeated**.

- Identifiers
  - **All classes** must have an identifier (otherwise are weak or specializations).
  - **Specialized** classes (subclasses) **do not have** identifier, since they inherit it from the superclass. They are not weak classes. They cannot redefine attributes.
  - **Association classes never** have identifiers.
  - A class never uses attributes referring to other class. We use associations

4.2. Conceptual Design

Example: Book library

We are interested in designing a database for managing a small library. After analyzing the system, the more frequent operations have been identified:

- Obtain the book information: book code, tittle, author (or authors), theme, and reader that has currently the book (if the book is currently borrowed).
- Obtain the information about the readers: reader id, name, address, phone number, and the books that the reader currently has and its loan date.
- Add, remove, and update the reader’s information.
- Manage the loans: Lend a book to a reader and record the return date.

Some integrity constraints have been identified:

- The book code uniquely identifies the book.
- The reader code uniquely identifies the reader.
- The themes to classify books are physics, electricity, mechanics, and optics.
- The total number of books borrowed by a reader is a derived attribute and it will be automatically updated by the system.

Example: Restaurant

“A restaurant wants to make their menus available through mobile devices. The information deals with menus, dishes, cooks, ingredients and wines. The restaurant offers several menus for which we have a code, a price and the dishes it is composed of (at least one dish in each menu). Each dish may be designed by a cook (but not more than one), with DNI (national identity number), name, country, and age. Each dish has a code, a name and (optionally) its approximate number of calories. For some dishes we also want to store information about its ingredients (including their quantity) and the recommended wine for the dish. We will consider the code of the wine, the type, the name and the vintage. Finally, for each ingredient we will store its code, name, price, and the stock. Having this information for each ingredient is compulsory.

Example: Cycling Race

A news website wants to create a new section to cover Le Tour de France, with complete results and statistics about the race. The readers will be able to get all the information about teams, cyclists (riders), climbs (mountain passes), jerseys (maillots) and stages. For each team we want to store the name, the coach and the racers ("cyclists") who compose the team. All cyclists must belong to one and only one team. For each cyclist we consider its number (assigned to the cyclist for the whole race), name, age, name of his team, stages he has won, climbs he has gone through on the first position and the jerseys he has worn in each stage. For each climb we consider the name, maximum height, category ("1st", "special",…), slope, stage where it is located (one and only one) and the cyclist who has passed it on first position. For each jersey, we will consider the jersey code, type, color, prize for wearing it and cyclists who have worn it.
For each stage we will consider stage number, length of the stage in km, departure city, arrival city and the cyclist who has won the stage.
Two different cyclists cannot wear the same jersey during the same stage.
Since the database will be updated during the race, many stages and climbs will have no winner initially. Information about who is wearing the jerseys for each stage is also unknown until each stage is finished.

Example: UPV Election

The “Universitat Politècnica de València” wants to create a database to manage the elections to UPV Rector. The census is composed of people belonging to the group of teachers, students, or administrative staff (PAS).

Each member of the census is identifies by a DNI (ID) and also a name, the group to which he/she belongs, and the center where she/he works. Each center is identified by a code and has a name and a size (in square meters).

There are several lists of candidates. Each list is identified by a unique code. We know the date of creation of the list and the members who compose the list. All member of a list must be in the census and we want to know the position of the member in the list of candidates. A member of the census cannot appear twice in a list or in more than one list.

The vote will take place in polling stations. Each polling station is identified in a center by a unique number (unique in that center). Each member of the census needs to know in which pooling station and center he will vote.

Four members of the census are the chairs of each polling station. One of them is the president of the polling station. A person can only be member of one polling station.

On a voluntary basis, a list of candidates may assign an auditor for each polling station. This auditor must be part of the census and can be auditor in several polling stations. An auditor cannot be auditor for more than one list of candidates.

Example: The UPV Sailing Club

This database will be used to contact crew members of the UPV (staff, students, and alumni) who want to participate in races, with boat owners looking for a crew for their boats.

For each boat we must store its main characteristics, such as model, name of the boat (that serves as id), length, beam, yacht club, and owner. We need to store for each owner his/her full name, DNI, mailing address, and one or more phone numbers (one at least). Moreover, the UPV Sailing Club has information on all the races: the name of the race, its organizing yacht club, and the race category. The races are celebrated each year (different editions). For each race edition we know the start and end dates, and the number of available positions on each boat. This information is transmitted to the members of the UPV Sailing Club to assign crew to each of the ships.

We need to create a final report containing the members of the club who have participated in each race edition (in a specific year) and which ship has been used. For each member of the club we need to store his/her sailing license number, his name and address, phone number, if he is a student (enrolled in a school) or a member of the staff, and his sailing qualifications. We will also store the basic contact information of the yacht clubs that organize the races: name, initials of its name (for identifying purpose), address, phone number, and a brief description.

Create a database about the different routes of the St. James's Way (Camino de Santiago). We want to keep information of the different routes (the French Way, the Aragonese Way, the Northern Way, etc.).

For each route we want to record its name, which is unique, its description and the number of total kilometers. Each route consists of a set of stages. A stage is identified by a sequential number within each route. For each stage we know the length in kilometers and some useful recommendations for the pilgrims.

Each stage runs through a set of locations. We must record this set of locations and the order in which they appear within the stage. We must be aware that a location can appear in several stages (of different routes) but a single route cannot pass twice through the same location.

Locations are identified by a code, and have a name. We also store their (touristic) interesting sites. For each site of interest, which is identified with a unique number within each location, we also store its name and a description.

For those locations (localities) having pilgrim's hostels we want to know the number of inhabitants and the hostels they have. A hostel is identified by a code, and has also a name, an address, and a price.

Some locations have one office where pilgrims can stamp the pilgrim's pass(port), which certifies that the pilgrim has been in that location. For each office we want to store its name and its full address. We also want to have a record of pilgrims with pass(port). Each pilgrim has an identifier, a name, and an address. We also record the locations she/he is certified to have visited the day of the visit.

Example: Supermarket

A large grocery store has decided to create a home-delivery service. The supermarket has prepared a warehouse next to the store where purchases are deposited until the dealer picks up and distribute them. In this warehouse several cold storages have been installed. Each cold storage is uniquely coded and we want to know its scheduled maintenance date and the name and telephone number of the maintenance technician. The store also has installed some numbered shelves. We are interested in knowing what the height of each shelf is. When a customer completes his/her purchase, if he is a new customer he must give his personal data (ID, name, phone, and address) to the cashier. The cashier is responsible for distributing the purchase in bags labeled with a serial number and entering this information into the system along with the ticket number (which identifies a purchase) and the date.

The bags can be labeled as "keep in cold place" or not. The bags are stored in the warehouse in shelves or in cold storage. The manager of the warehouse saves where each bag is located. The delivery boy goes to the warehouse and pick up the ticket of a purchase. The ticket includes the position of all the bags of that purchase and who was the cashier that prepared this purchase.

It is very important to know the time that each purchase has been delivered and the delivery boy who delivered that purchase. This information is introduced by the delivery boy and will be used to calculate a gratification for the workers at the end of the month.

The system must store the worker id, name, age and phone number of each delivery boy. The worker id, name, and category of each cashier will be also stored.

Example: A building company

A large construction company dedicated to the refurbishment of properties wants to create a database. We will store the employee number, which is unique, his/her DNI (which is also unique) the name, address, telephone and his/her role in the company (architect, architectural technician or worker).
When a client requests a reform, all his/her data is stored: DNI (his identification), name, full address and contact phone, and the data related to the refurbishment: a unique code, the refurbishment and the full address of the property to be refurbished. A refurbishment is assigned to an architect, who will be in charge of making the refurbishment project. We also know the architect collegiate number (which is unique) and his specialty.
When the architect prepares a refurbishment project, he/she marks the refurbishment as ready, a copy of the project is registered, and an expected date of the refurbishment is registered.
When a refurbish starts, it is recorded as initiated; we store the start date and the architectural technician who will be responsible for the supervision. We have to store the architectural technician mobile number in the database. Every day we have to generate a report for any active refurbishment. In this report must appear the workers who will be working that day, and for each of them, the task assigned. The works to be performed by the workers are classified and have a unique code and description. We have to store for each worker the tasks for which he/she is qualified. A worker may be assigned only a task for which he is qualified. Moreover, the daily report also includes a list of the materials to be used that day, and the amount of each one of them. There is a record of the materials (with a unique code), a description, and their stocks.
At the end of the day, each worker must tell to the architectural technician the incidences occurred in the tasks that he was assigned. This information is recorded in the daily report.
When a refurbishment is finished the ending date is stored in the database.

Example: Traffic

Nopacekistan (a country) is divided into traffic zones known as sections. The country's traffic police want to create an information system to have a better control of traffic fines and accidents. In order to do so, the police will store information about citizens, including their identifier, name and address.

The traffic police unit is composed of officers who always belong to one section. For each officer we know their collar number, which identifies them, their name and address.

For each section we will have an identifier, the number of inhabitants, the vehicles that pay their taxes inside that section and, finally, some information about fines and accidents.

Every fine and every accident always takes place in one section.

Fines are identified (within a section) by a number. Accidents are coded with a sequential number, re-initialised every day.

For each accident, we will have its date, time and the place where it happened, the officer that handled it and the vehicles (at least one) that were involved, also indicating the state of each vehicle after the accident. Each vehicle's driver will also be stored whenever possible.

Fines are issued to a specific vehicle, also specifying the place, the kind of penalty, the amount and the officer who issued the fine.

Finally, each vehicle is identified by its number of plate, and has a vehicle type (car, motorcycle, …), model and colour.

Example: Pizzeria

The pizza restaurant “Buona” has started the development of an information system for home delivery and, after a first analysis stage, the following information requirements have been identified.

The restaurant hires employees for whom we know their DNI (national identifier number), name, surnames and category. The category of the employee can be cook, assistant or delivery person. We want to store the certification for the cooks and the driving licence types for delivery persons (they must have at least one valid driving licence type).

The restaurant has a fleet of delivery vehicles that are identified by their licence plate, for which we may possibly know their model, and the date of the upcoming revision by the Ministry of Transport (ITV/MOT test).

The restaurant prepares several pizzas. Each pizza has a code that identifies it, its name and its price. In addition, we want to store the ingredients that are needed for their preparation.

Each ingredient has a reference number, which is unique for each supplier, and also has an ingredient name and its price. In addition, for each ingredient we want to know the remaining amount in stock. Suppliers can be identified by their NIF (tax identification number). Suppliers also have a name and, frequently, they also have a telephone number.

Each customer placing an order is identified by their DNI. We also store their name, surnames, address and their telephone numbers (line and/or mobile, at least one). Whenever an order is received, we assign a code that identifies it (which is unique for the day), the date and the time of the order, the customer that places it and the content of the order (composed of the code of each ordered pizza and the quantity) and the total cost of the order, which will be calculated automatically from the order data. Later on, the order will be assigned to one or more delivery persons (depending on the size of the order), each of them taking an available vehicle in order to perform the delivery. We must store the vehicle that each delivery person has used for the delivery and the time it has been delivered (this information will be added to the system when the delivery person is back to the restaurant).