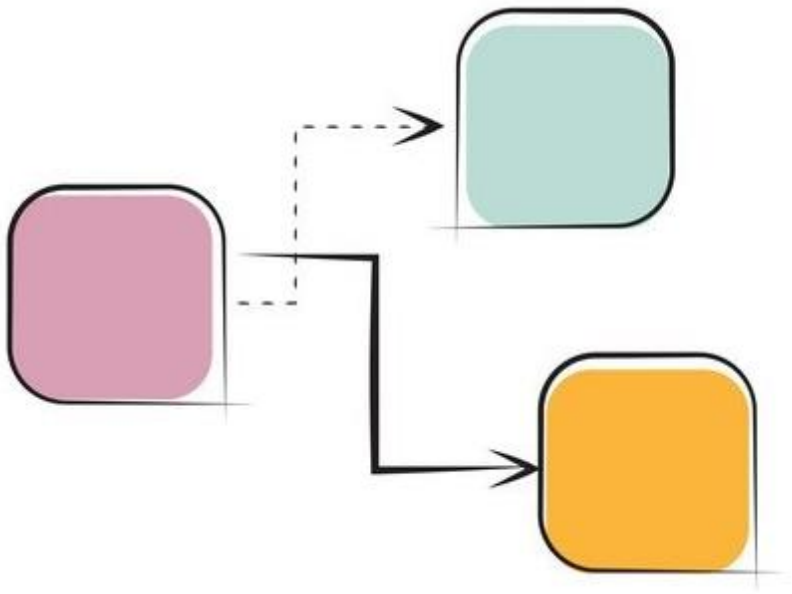
Universitat Oberta de Catalunya

Software Design Patterns

Preliminary exercises

Compilation of exercises on the main UML diagrams.



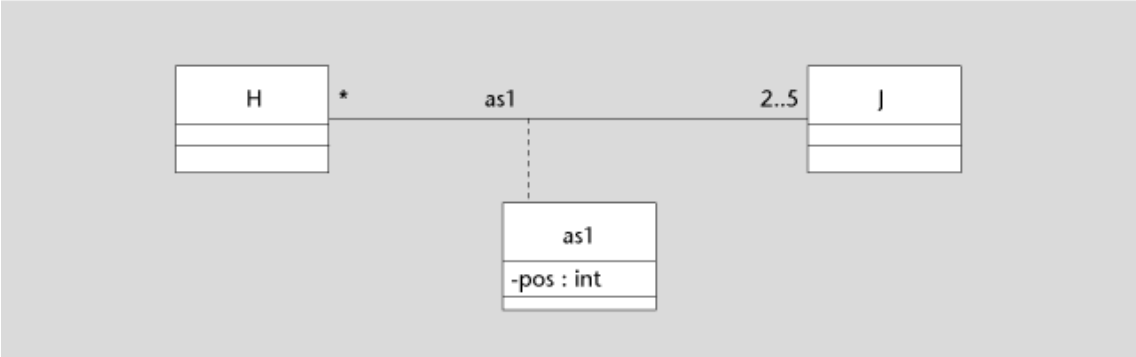
Software Design Patterns teaching team
Last updated: September 2020

Exercise 1: UML notation

UML (Unified Modeling Language) is the standard language to create graphical software models.

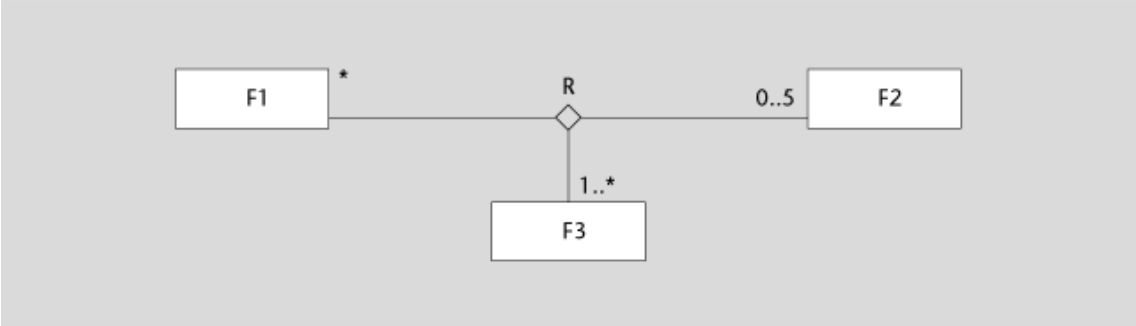
Represent, using a UML class diagram, the following situations:

a) We have two classes: *H* and *J* associated through the *as1* association. Each instance of *H* may be associated with at least two and at most five instances of *J*. Each instance of *J* may be associated with an unspecified number of instances of *H*. Each time an instance of *H* and *J* are associated, we want to know what the relative position of that instance of *J is* among all those associated with *H* (we will use an attribute of integer type called *pos*).

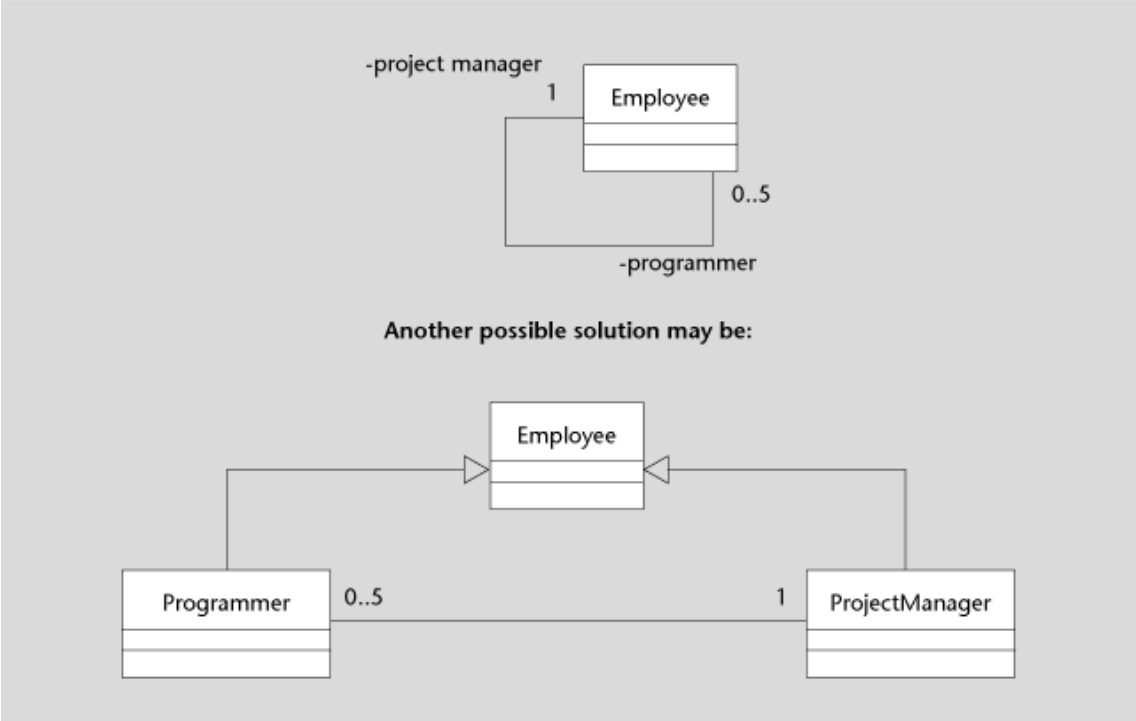


b) We have two classes: *K* and *L*. Each instance of *L* belongs to an instance of *K* to the extent that if we delete its instance of *K* from the system, we will also delete the instance of *L*. Each instance of *K* can have a maximum of nineteen instances of *L*.

c) We define a ternary association *R* between classes *F1*, *F2* and *F3*. Any pair of instances of the classes *F1* and *F2* must be associated with at least one instance of *F3*. For each pair of instances of *F1* and *F3*, there are a maximum of five instances of *F2*. For the pair of instances *F2* and *F3*, there is no restriction on the cardinality of *R*.



d) Each employee of a software company can have the role of programmer or project manager. In the role of project manager, he or she can lead up to five programmers, or none. Each programmer will have only one project manager.



Note that if we consider that an employee can be a programmer and a project manager at the same time, then this solution implies the existence of multiple classification. Multiple classification allows an object to be an instance of both the class *Programmer* and the class *ProjectManager*; OO programming languages do not allow this to happen, so in the design stage this class diagram should be transformed using appropriate design patterns.

Exercise 2: Conceptual data model (static analysis diagram)

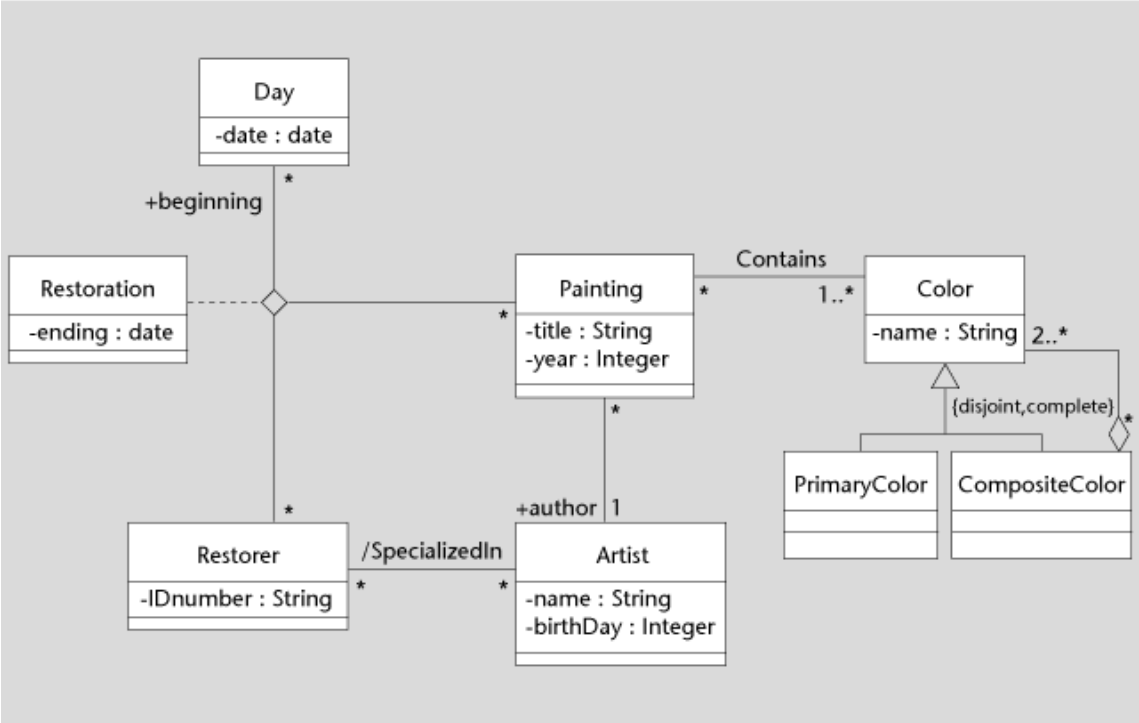
The class diagram, also called conceptual data model or static diagram, is probably the most well-known UML diagram. It is used to model classes and their associations, emphasizing the structural aspects rather than their behavior.

We want to create a system that enables the control of the painting restorations that are made in a museum. We want to know the title, the year in which it was painted and the author of each painting. The paintings are identified by the name of the painting and the name of the artist. Regarding the authors, we want to know their name and birth year.

Each painting contains a set of colors, whose names we want to know, which can be primary or composite. Regarding composite colors, we want to know which are the colors that form them (that can be at the same time composite).

Each panting can be restored several times and by several restorers. We want to know the ID number of each restorer, as well as who are the artists he or she specializes in, which are the artists from whom he or she has restored more than one painting. A restorer can be involved in several restorations of the same painting. We want to know the beginning and ending dates of each involvement. It may happen that a restorer is working on different paintings on the same date. On the other hand, at a given date several restorers may be working on the same painting.

Create the corresponding class diagram.



NOTE: In both attributes and association roles, visibilities should not appear as this is an analysis diagram. They are present due to a limitation of the tool.

NOTE: Note that the beginning date should be involved in the *Restoration* association because otherwise, a restorer could not restore the same painting twice (which should be allowed, according to the description of the exercise). The reason is that, by the definition of association, there cannot be two instances of an association with the same objects, that is, if we did not make the beginning date to be involved in the ternary, there could not be two instances of *Restoration* with the same *Painting* and *Restorer*. To allow instances

with these two extremes repeated, we must add the class *Day* as a participant in the association. The ending date, on the other hand, is enough for it to be an attribute of the class *Restoration*.

NOTE: The specification of all restrictions and derivation rules that cannot be expressed graphically is missing.

For example:

- A restorer specializes in an artist when there is more than one *Restoration* instance that relates a restorer to an artist's work (*SpecializedIn* derived association).
- There cannot be two instances of *Painting* with the same name if they are associated with the same artist (uniqueness restriction of *Painting*).

Exercise 3: Use case diagram

The use case diagram describes the relationships and dependencies between the *use cases* and the actors involved. The use case diagram complements, but does not replace, the textual description of use cases, as it does not include information on the behavior of the system.

The objective is to automate the city council police management, where the following positions exist:

The Corporal, who is assigned with the task to:

- Register police officers in the system. Personal information of each police officer such as surname and name, ID number, gender, birth date, a text on training received, and another on previous work experiences must be stored.
- Create the daily report of the car patrols, with the date of the day and the first annotation of the day. Annotations have a correlative number, the surname and name of the author and the text.

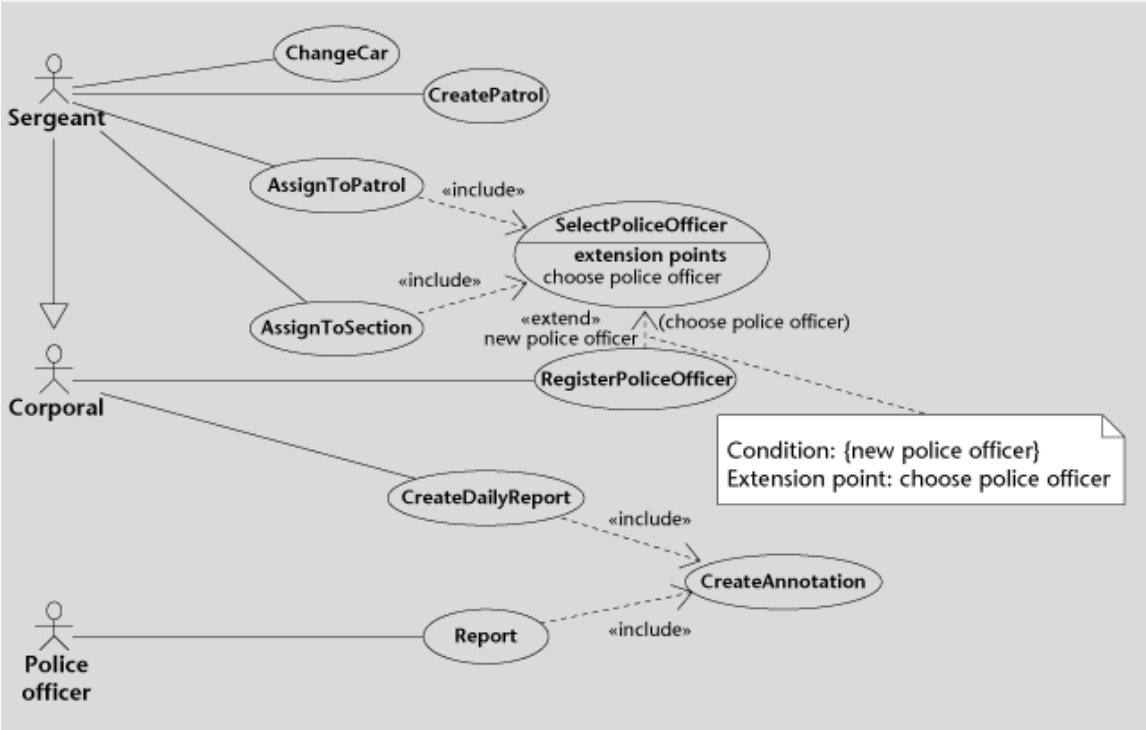
The Sergeant, who is assigned with the task to:

- Create patrols. Each patrol has a code and can be associated with a car or not. When a car patrol is created, the Sergeant assigns a car, which is selected from a list of available car license plates. He or she can also change the car assigned to a patrol.
- Assign tasks to police officers; two types of tasks can be assigned: assign an officer to a patrol, or assign an officer to a section, which are two separate functions. Patrols can have up to four police officers. When a police officer is assigned to a section, the role he or she will play must be described. Each police officer assigned to a task –either a patrol or a section– is either chosen from a list of police officers surnames and names, or registered at that moment. A police officer may be assigned to only one task; therefore, when a task is assigned, the previous assignment, if any, is deleted.
- The Sergeant can also perform the functions of the Corporal.

The police officer, who is assigned with the task to:

- Report. To report, the police officer must introduce his or her ID number, the patrol code and the license plate of the car assigned to it and then create an annotation, the number of which will be assigned in a correlative way, and the system will get the surname and name of the author from the ID number introduced. The annotation text will then be created by the police officer.

a) Create the use case diagram.



b) Create the textual specification of the use cases involved in the assignment of a police officer to a task, broken down in numbered steps, specifying in each whether it is done by the system or by an actor, and the data involved.

Use case: SelectPoliceOfficer
Functionality summary: To select a police officer.
Actors: Sergeant
Related use cases: -
Precondition: -
Postcondition: A valid police officer has been selected on the system.
Main scenario:
1) The system displays the surnames and names of the police officers in the system, and gives the Sergeant the option to select one.
Extensions: The Sergeant selects one of the police officers.

Use case: AssignToPatrol
Functionality summary: To assign a police officer to a patrol.
Actors: Sergeant
Related use cases: SelectPoliceOfficer
Precondition: There is at least one patrol with less than four police officers assigned.
Postcondition: A police officer has been assigned to the patrol.
Main scenario:
1) The Sergeant selects a police officer (include SelectPoliceOfficer).
2) The system displays the code of patrols with fewer than four police officers and gives the Sergeant the option to select one.
3) The Sergeant selects one of the patrols displayed and confirms the selection.
4) The system deletes from the database the previous assignment of the police officer to a section or patrol.
5) The system stores in the database the assignment of the police officer to the selected patrol.

Process alternatives and exceptions:
2a. There is no patrol with less than four police officers:
2a1. The system cancels the use case.

Use case: AssignToSection
Functionality summary: To assign a police officer to a section.
Actors: Sergeant
Related use cases: SelectPoliceOfficer
Precondition: There is at least one section.
Postcondition: A police officer has been assigned to a section.
Main scenario:
1) The Sergeant selects a police officer (include SelectPoliceOfficer).
2) The system displays the name of all the sections and gives the Sergeant the option to select one and to introduce the function.
3) The Sergeant selects one of the sections displayed, introduces the function and confirms the selection.
4) The system deletes from the database the previous assignment of the police officer to a section or patrol.
5) The system saves the assignment of the police officer to the selected section with the function introduced.
Process alternatives and exceptions: None

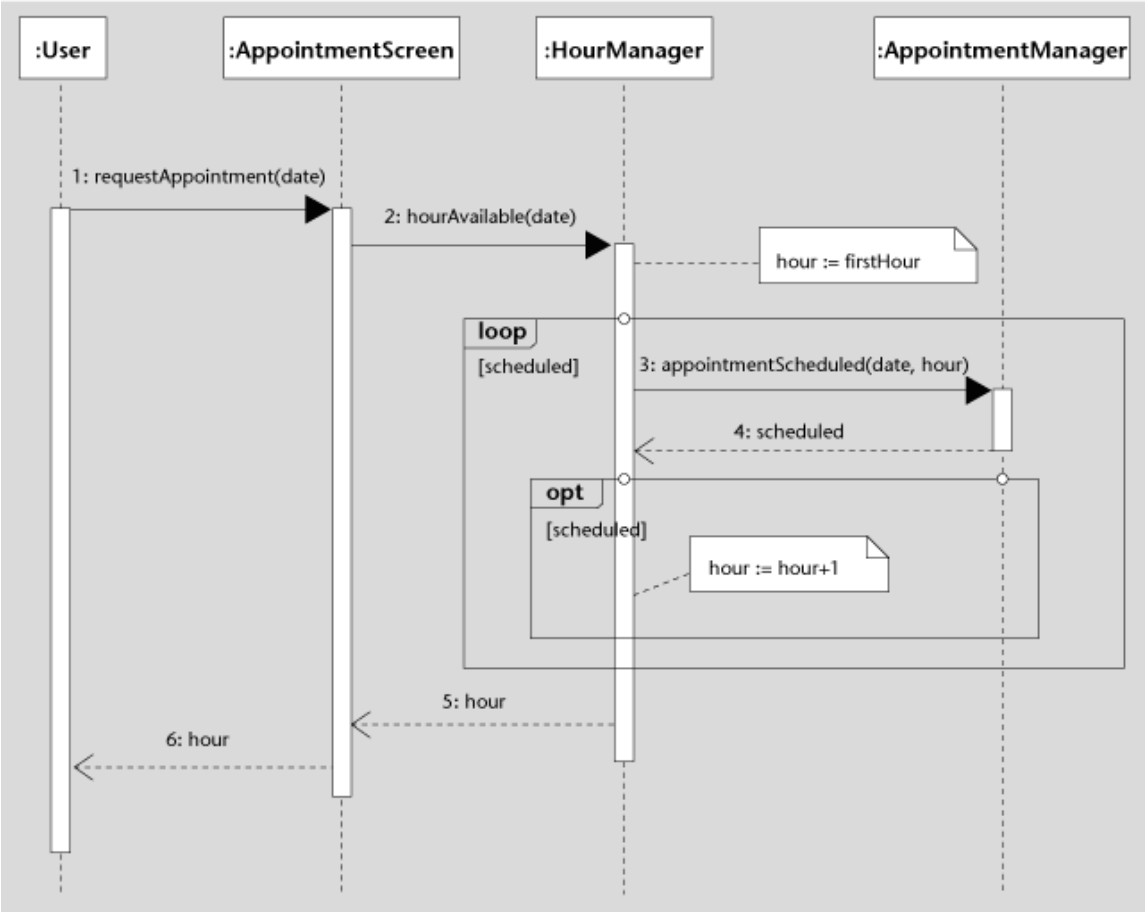
Use case: RegisterPoliceOfficer
Functionality summary: To add a police officer to the database.
Actors: Corporal and, by inheritance, Sergeant
Related use cases: SelectPoliceOfficer
Precondition: -
Postcondition: A new police officer has been registered.
Main scenario:
1) The system asks the Corporal for the police officer information: surname and name, ID number, gender, birth date, training received, and previous working experience.
2) The Corporal introduces the data requested and confirms them.
3) The system creates a police officer with the data introduced in the database.
Process alternatives and exceptions: None

Exercise 4: Sequence diagram

The sequence diagram simulates the interaction between various objects, emphasizing the temporal sequence (in which order the objects interact).

We want to design a patient appointment management system for a medical center. When a user requests an appointment, he or she indicates, on the appointment screen, the date they want the appointment for. This screen will call the *hourAvailable(date)* operation of the time manager instance which, in turn, will do the following: it will go to the appointment manager to ask whether there is already an appointment scheduled at the first hour or not. If a visit is already scheduled, it will try the next hour until it finds an hour for which there is no scheduled visit on that date.

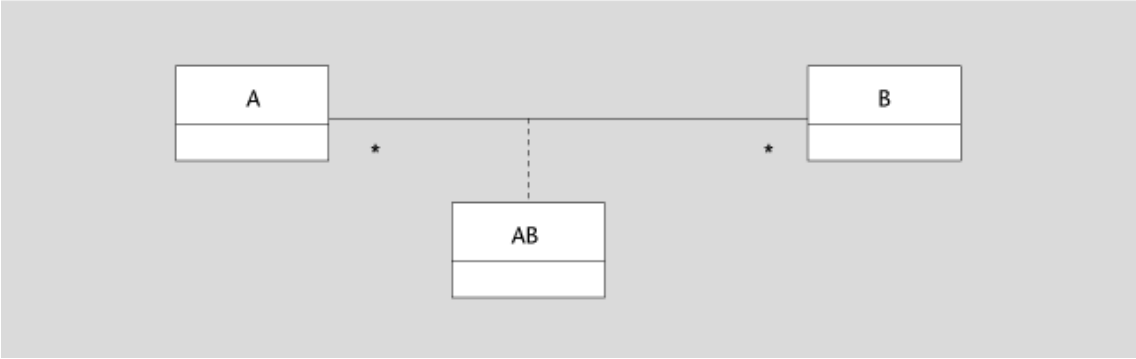
Create a sequence diagram representing the situation described.



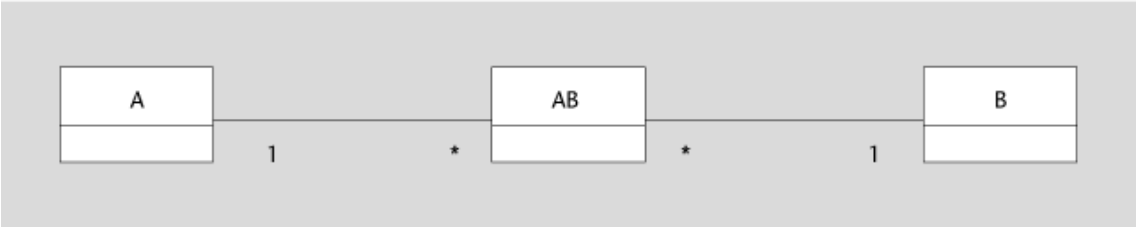
Exercise 5: Normalization of association classes

The concept of association class does not exist in programming languages, so association classes must be converted to standard classes. This process is known as normalization or reification of association classes.

Given the following model:



Represent with a graphic an equivalent model using only binary associations one to * and without using any association class.



Integrity constraint: An instance of AB may only be associated once with the same pair of instances of A and B.

* The restriction has been textually expressed since UML does not have graphical notation to do this. What could actually be done is to formally define it using the OCL language (Object Constraint Language).