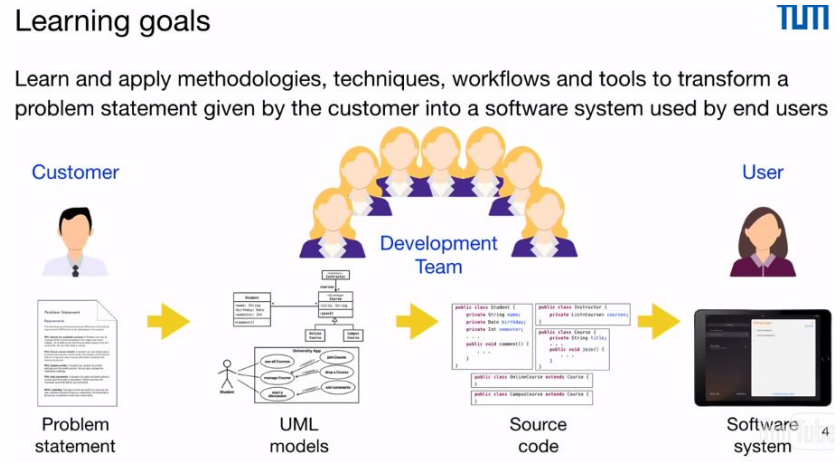


1. Class Definition

1.1. **Fuente.** KULeuvenX: UMLx UML Class Diagrams for Software Engineering (Copia Textual - Literature Review).

1.1.1. UML Class Diagrams for Software Engineering

1.1.2. Welcome

1.1.3. Modelling languages

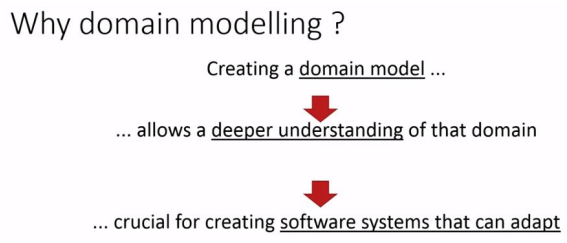
1.1.4. A class is an abstraction

1.1.5. Why are class definitions important?

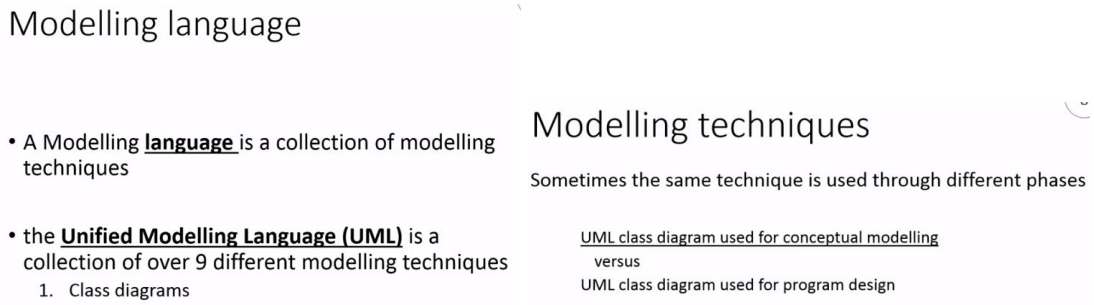
The Unified Modelling Language was first proposed as a standard in 1996. Since then, UML has become the most widely used language to draw up plans for software systems. It is used by software engineers, requirements engineers and business analysts. Knowledge of UML has become a much sought-after skill in software development and engineering. Some of today's top jobs require UML knowledge. Think about business analysts, enterprise architects, but also developers, technical consultants and solutions architects...In this course we'll focus on UML Class diagrams and their use for **conceptual data modelling**... we look at the essential aspects of conceptual domain modelling.

"Dictionary.com" defines a **model** as "A systematic description of an object or phenomenon that shares important characteristics with the object or phenomenon." So, models present a systematic, and most often simplified description of what they represent.

Each business is unique and will, therefore, yield a different domain model, adapted to its specific needs..UML class diagrams is the prime modeling instrument used to map out specific business domains. Creating a UML class diagram for a business will enable to pinpoint what makes this business unique and will provide a good starting point for developing an information system.

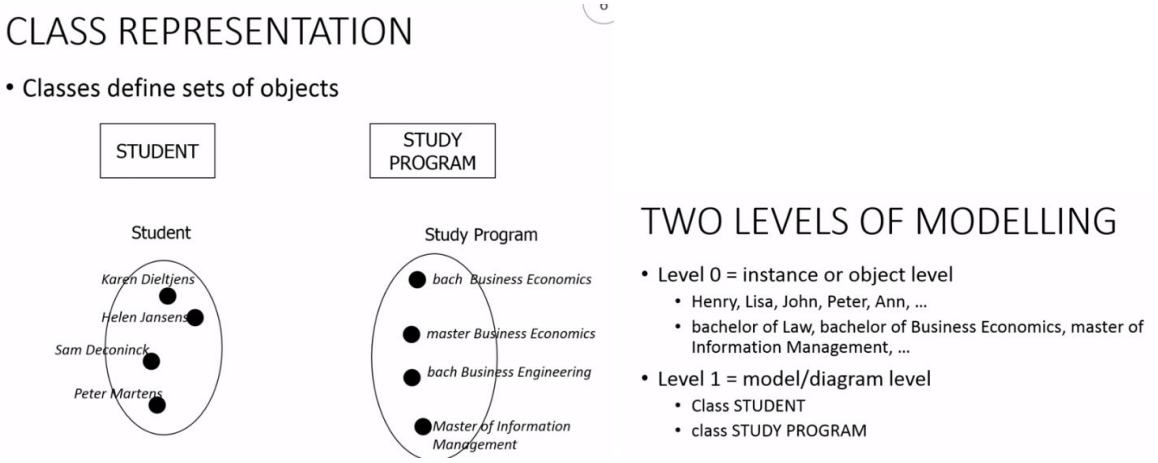


Creating a domain model allows a deeper understanding of that domain. And, this deeper understanding is crucial for creating software systems that can adapt to the ever changing requirements of their users.,,So, the lesson to remember is that you need to understand the core concept of a business. The domain model that captures these core concepts need to be at the heart of your information system to ensure adaptability.

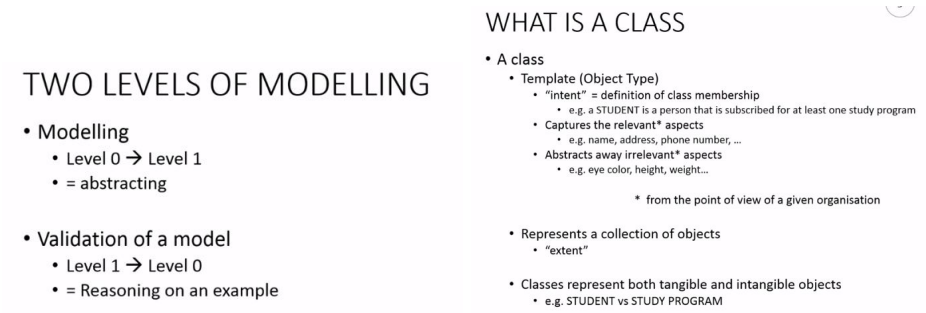


Class definition

In computer terms, a concept that groups objects with similar characteristics is called a 'Class'... in order to be able to handle the objects in the universe of an organisation, these objects need to be categorized into concepts. Example: University has the following classes: Student and Study Program



Modelling is abstracting, it means that you make the transition from level 0 to level 1. You go from the instances to the definition of the concepts. On the other hand when you want to validate the model you have to do the reverse. So you will take the model and reason on an example.



A 'Class' has two functions.

1. On the one hand, the class will be **the template** or a model for a group of real world objects that are similar. It defines a type of instances and therefore we also call it an object type. This is called the **"intent"** of the class; it is the definition of the concept that defines class membership. The template will **capture characteristics that are relevant** about the objects in the class. **The class definition will omit the irrelevant aspects.**
2. At the same time, the class represents a collection of objects that conform to its intent. This is called the **'extent'** of the class. Notice that classes can represent both tangible and intangible objects.

2. Attributes and Data Type

2.1. **Fuente.** KULeuvenX: UMLx UML Class Diagrams for Software Engineering (Copia Textual - Literature Review).

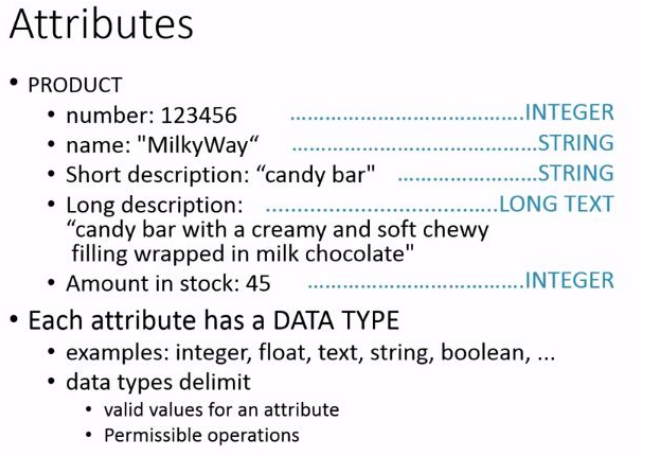
2.1.1. Attributes and Data types

2.1.2. Respresenting Instances in MSAccess or MSExcel

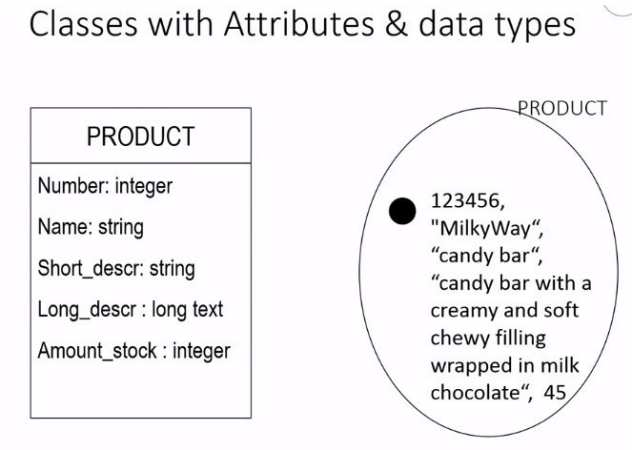
Attributes / Data Types

The characteristics that we want to information about for every individual object in a class, are called the **'attributes'** of that class. The list of attributes for a class is obviously specific for a particular domain.

For a university, the mentioned attributes in the figure are relevant. Each object has its own values for the attributes: attribute values are specific for the instance they belong to. Moreover each attribute has a data type. Examples of data types are integer, float, text, string, and boolean. The data type constraints the values that can be given for a particular attribute.



When an instance is given a value for an attribute, this value has to follow the constraints implied by the data type. By the fact that the data types constrain the permissible values for an attribute, they also define the permissible operations. In a UML class diagram the attributes and their data types are written in a box below the class's name,



3. Associations

3.1. **Fuente.** KULeuvenX: UMLx UML Class Diagrams for Software Engineering (Copia Textual - Literature Review).

3.1.1. Association Basics

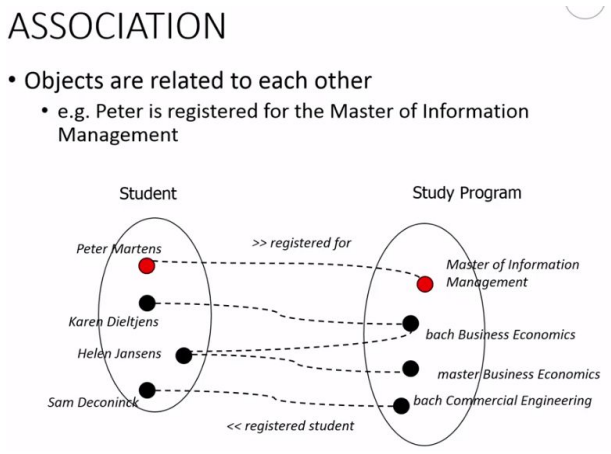
3.1.2. Unary association

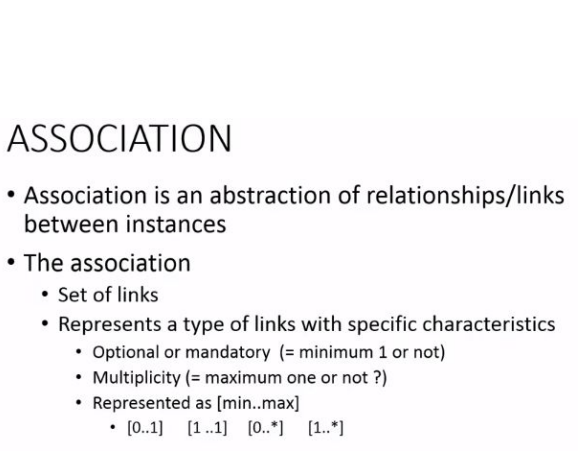
3.1.3. Ternary association

3.1.4. Aggregation

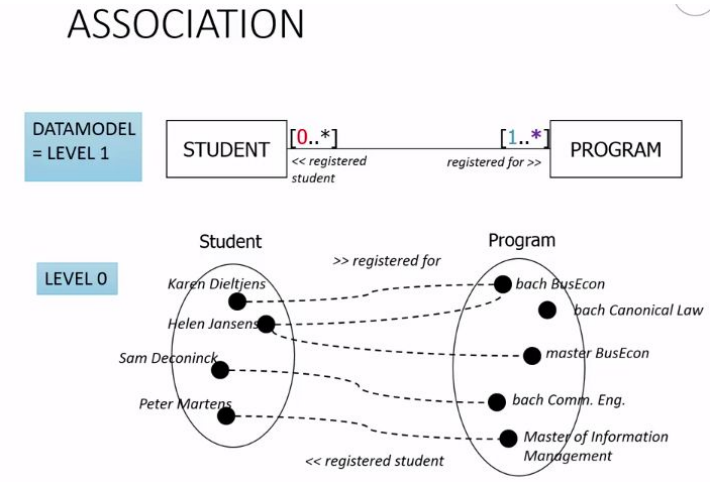
Associations

Following the principle of abstraction means that **similar relationships or links between instances at level 0 need to be abstracted into a higher level concept at level 1**. **Such abstract concept is called an association**. An Association is therefore a level 1 concept that relates classes and that represents a collection of links between the objects of those classes. **It represents a type of link with specific characteristics**. The first of these characteristics is whether it is optional or mandatory to have a link with other objects. So for example is it optional or mandatory for a student to be registered for a program? The other important characteristic is the maximum number of objects we will find at the other side of the association. In this example maximum number of programs a student can be subscribed to at a any single point in time? The combination of the minimum and maximum is represented as an interval: minimum .. maximum. The minimum can be 0 or 1, and the maximum can be 1 or many and the many is represented as a star:



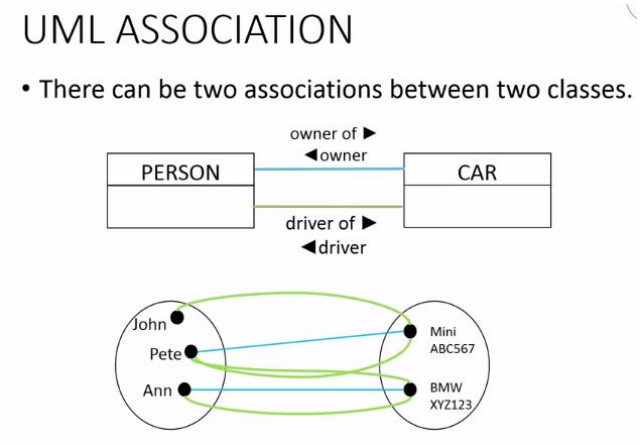


Let's look at the following association example as we must remember the rule ***Modelling is abstracting, it means that you make the transition from level 0 to level 1.*** *You go from the instances to the definition of the concepts. On the other hand when you want to validate the model you have to do the reverse. So you will take the model and reason on an example*:

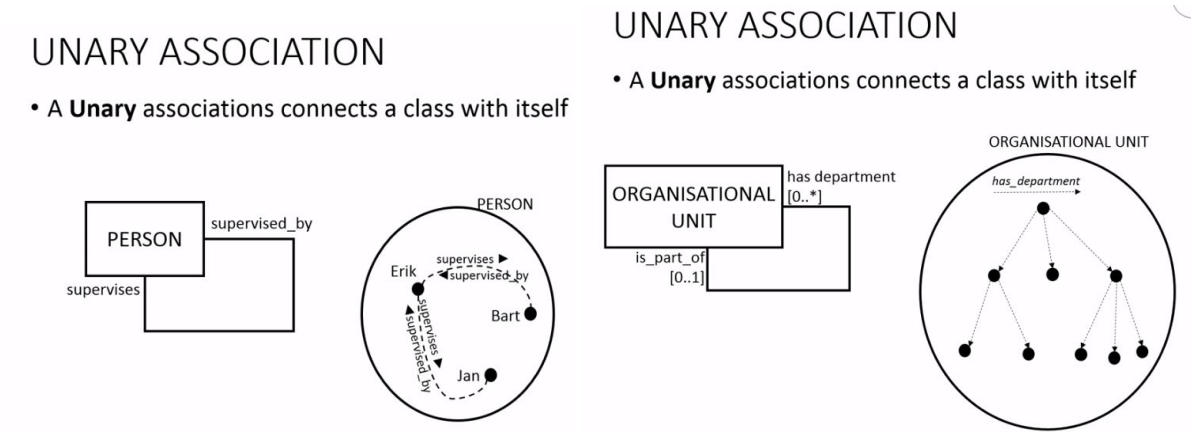


An association can be read in two directions. here: from the student to the program, and from the program to the student. These individual directions are called the **'roles'** of the association. In this example, the role from the student to the program has been named 'Registered for' and the role name is put next to the destination class of the role, so next to the class Program. In the other direction a program has registered students, so we put the role name 'registered students' next to the class Student.

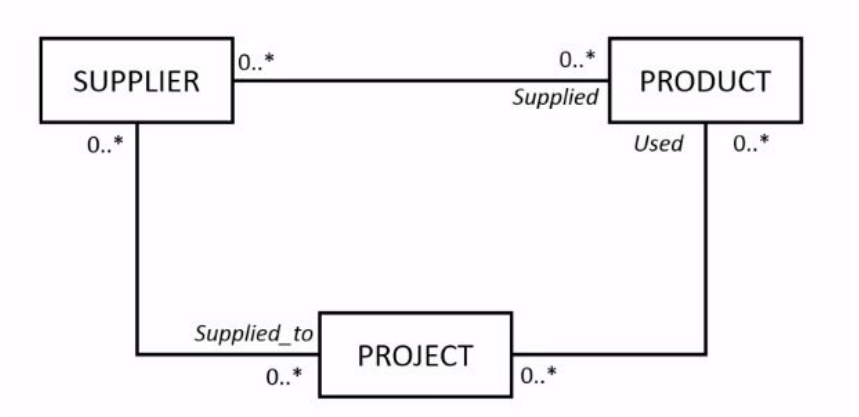
When you have two classes in a UML class diagram you can have two different associations between those two same classes.



In some cases an association connects a class to itself. This is called a **unary association**. An example is when people are related to each other.



Sometimes binary and unary associations are not able to correctly capture the relationships between objects. Consider the following example. Assume we have Suppliers, Products and Projects, and we want to capture which supplier supplied what construction material for which construction project.



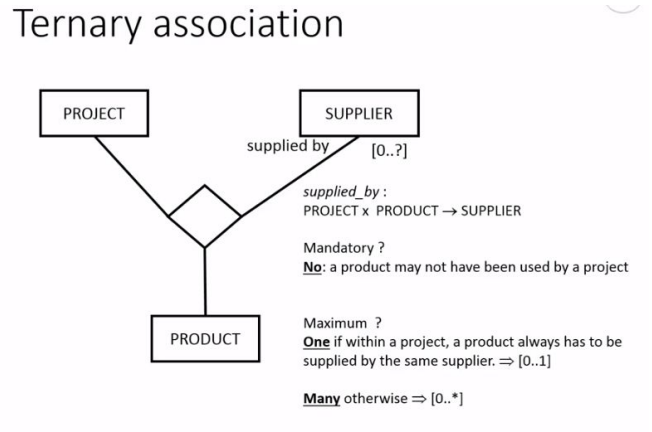
supplied	
SUPPLIER	PRODUCT
Johnson	Tiles
Johnson	Bricks
Peters	Concrete
MaxConstruct	Concrete
MaxConstruct	Wooden beams
Peters	Wooden beams

Supplied_to	
SUPPLIER	PROJECT
Johnson	P1
Johnson	P2
Peters	P2
MaxConstruct	P2
MaxConstruct	P3

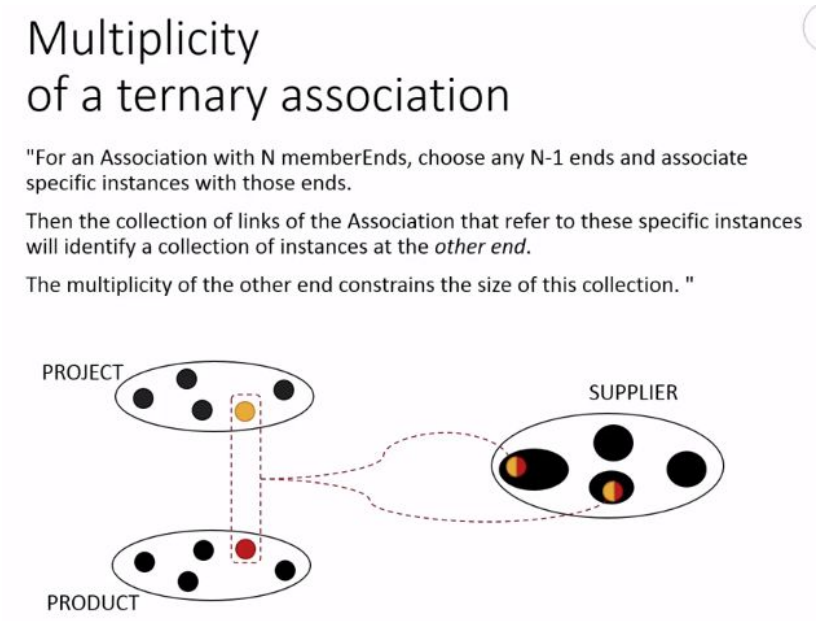
used	
PROJECT	PRODUCT
P1	Tiles
P1	Bricks
P2	Tiles
P2	Bricks
P2	Concrete
P3	Wooden beam
P2	Wooden beam
P3	Concrete

Who supplied the concrete for project P2?

So the combination of the three tables just doesn't give you enough information to answer that question. The only way to correctly capture which supplier supplied which product for which project, is to use a three-way association. like this, also called a **ternary association**.



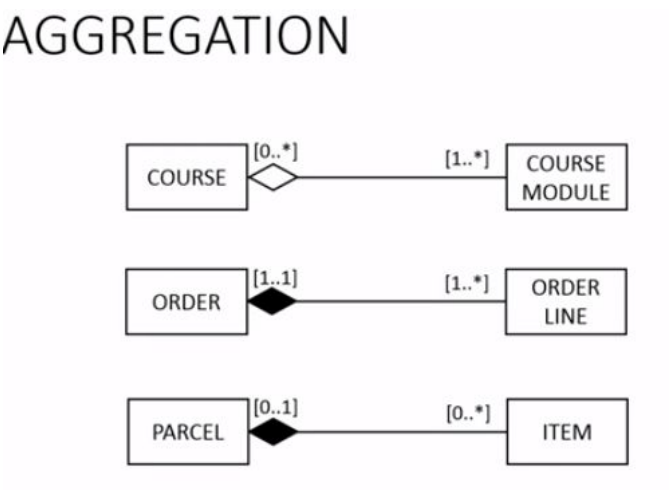
The interpretation of cardinality constraints for a ternary association requires however some care. If you put a zero to many cardinality next to supplier, what does it mean? Does it mean that each product and each project each can have zero to many suppliers? This is what the UML manual says about the interpretation of multiplicities: Using the same example:



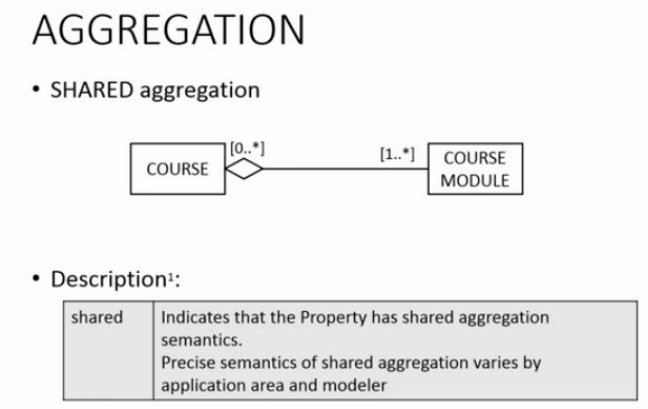
So, to determine the multiplicity on the side of supplier, you take a pair of one project and one product. And then you look at the set of suppliers that match with this (project, product) pair. The (project, product) pair could have been sourced from no, one or several suppliers. If the product has not been used in that project, you won't find any supplier. If it has been supplied by only one supplier, then you'll find one supplier. If the product has been sourced from two or more, then you'll find many suppliers. You have to do this for all project-product pairs to determine the general rule that is applicable. If your model allows to distinguish, for example, different orders at the same supplier, then a (project, product) pair may match several times with the same supplier. In such case additional identifying attributes will be required to distinguish the different links from each other. We'll come back to this when discussing the concept of AssociationClass. So for this example, in order to determine what to write next to the "supplied_by" role, we need to consider all project, product pairs, and see how many suppliers we find. The role will most likely not be mandatory: some products will not be used by all projects. For those pairs, we won't find any suppliers. So the minimum is 0. For the maximum, it depends on the rules. If within a project, a

product always has to be supplied by the same supplier, then the maximum is one. If you can source a product from many suppliers within the context of a single project, it may be many.

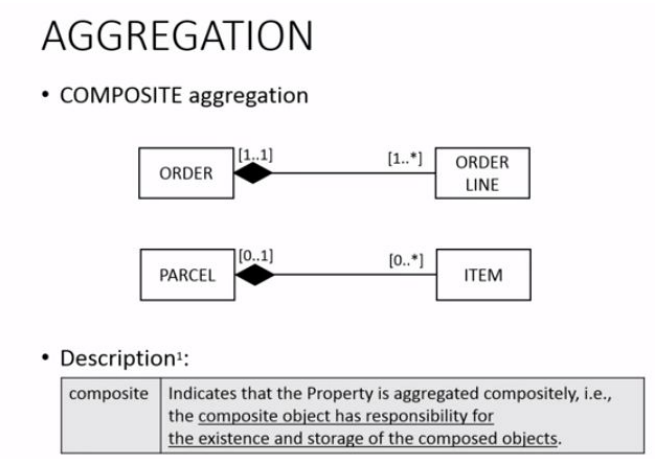
Some associations convey a meaning of a "part of" relation or composition. For example, this course is "composed of" a number of modules, an orderline is "part of" an order, a parcel is "composed of" items, and so on. In the UML, you can adorn the association end at the side of the "whole" with a diamond to clarify that the association has a "whole-part" meaning.



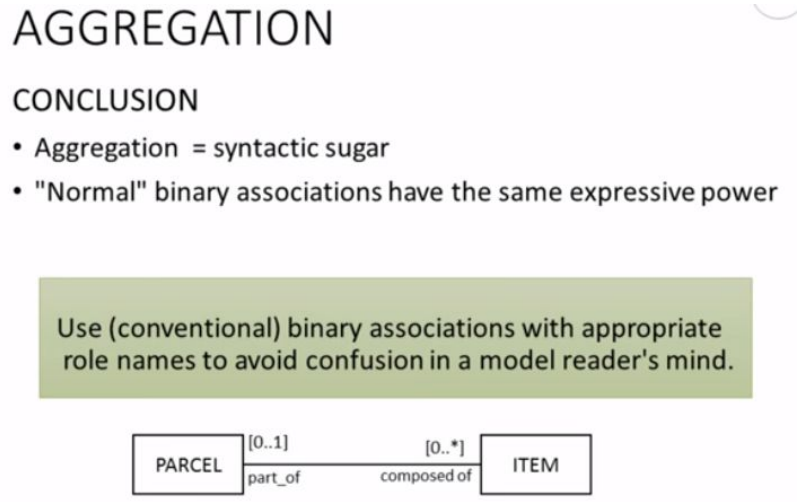
The diamond comes with two flavours: a white diamond representing a "shared" aggregation and a black. The white diamond represents a shared aggregation. Intuitively, this means that the parts can be shared by different wholes. If course modules can be shared across courses, then this would be an example of a shared aggregation.



The black diamond represents a composite aggregation, which is intended as a stronger form of ownership, meaning that the parts are owned by only one whole. The order lines being part of an order, and the items being part of a parcel are examples of a composite aggregation.



So the advice for good modelling is to use conventional binary associations with appropriate role names to avoid confusion in a model reader's mind.



4. Implicit Associations - Navigation and Parallel Paths

4.1. **Fuente.** KULeuvenX: UMLx UML Class Diagrams for Software Engineering (Copia Textual - Literature Review).

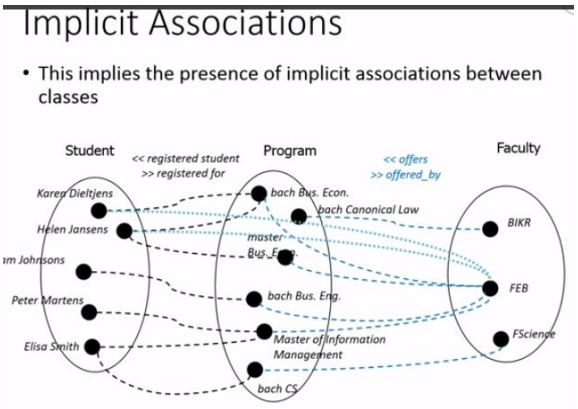
4.1.1. Derived/implicit association.

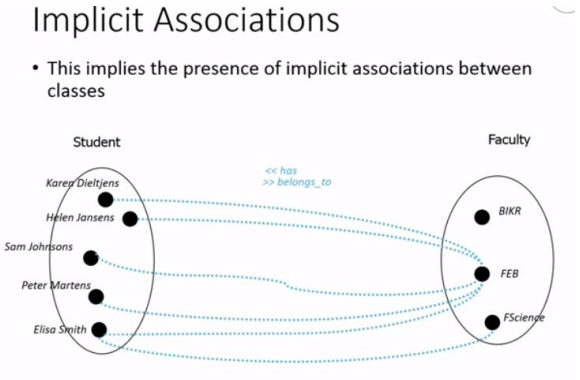
4.1.2. Navigation & information need satisfaction

4.1.3. Parallel paths

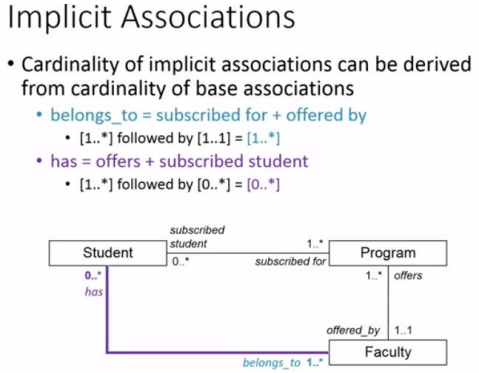
Implicit Associations

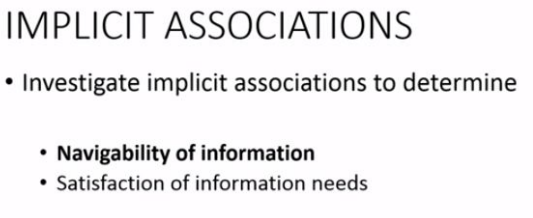
The fact that you can navigate over several consecutive associations implies that there are implicit associations between classes. In this example, there is an implicit association from the student to faculty that represents to which faculty a student belongs and which students a faculty has.





So, we have an association between students and faculty that represents which faculty a student belongs to and which students a faculty has. Note that such implicit association is not drawn on the diagram. But you have to be aware of its existence while reading a diagram.

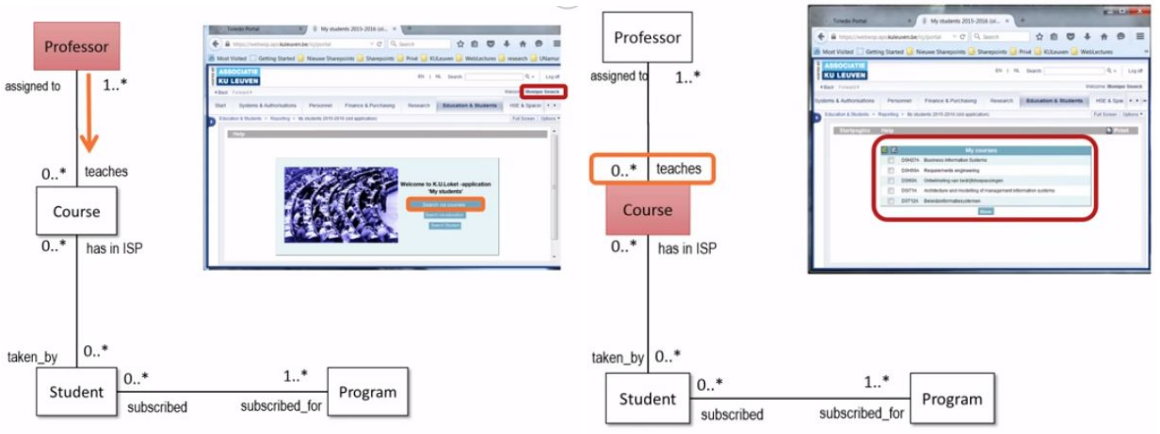




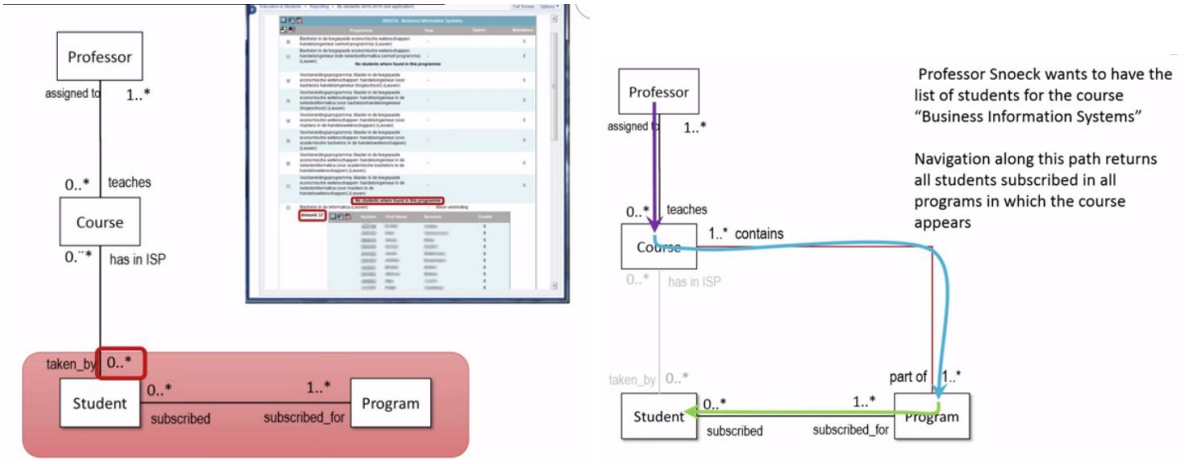
Navigation

By connecting multiple associations in a chain we can navigate from one end of the model to another. This navigability of associations determines the navigability of information in an information system: when applications are built based on a UML class diagram the associations in that diagram will define how in an application a user can navigate from one piece of information to another. As a result of this, the navigation paths will also impact the possibilities to satisfy information needs.

Let's look at an example. I have logged into the ERP system of my university and this means that in the class professor one instance has been selected, namely me. Now what I want to do in the ERP application is to find the students I'm teaching to. In order to navigate to my students the first thing I will have to do is to navigate to my courses. In the ERP-system I have a button 'Search via courses'. This corresponds to navigating in the UML diagram from the professor to the class courses:



Now, I will select one course: I choose for the first one, the "business information systems" course. By clicking the 'show' button, I will navigate to the students who follow that course. And what do I see? The application offers me a list of students but not just a flat list. It will group the students according to the program these students are subscribed to. At the right appears a version which returns a wrong answer to the question the professor asked.



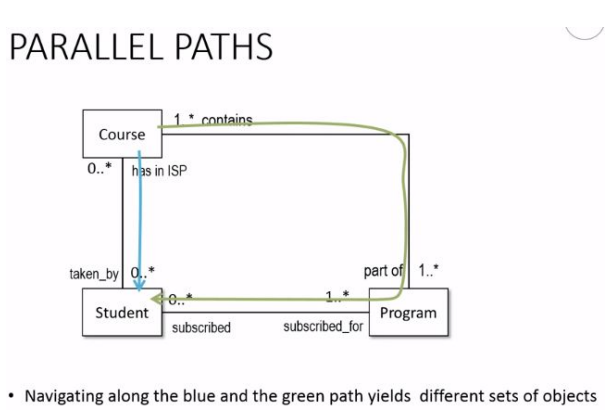
So, we see that clearly the non-availability of the grey link going directly from course to students hampers the satisfaction of the need to know how many students are subscribed in the course business information systems. So, wrong associations and missing associations are detrimental to information need satisfaction.

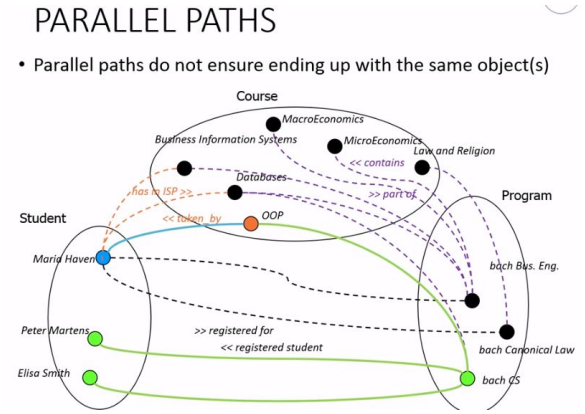
Parallel Path

An important thing we have to take into account when reading a UML class diagram, is that –based on our domain knowledge- we may assume that certain 'intuitive' constraints are imposed, but which in fact are not part of the UML class diagram. This is particularly the case when a UML class diagram contains parallel paths to navigate from one class to another.

For example, here we can see that we can navigate along the blue path from course to students. This gives us the students that are subscribed to a course. We can also navigate along the green path. In the first leg of this path, we navigate from course to program and we find all the programs the course is a part of. In the second leg, we navigate from program to student, and so we find all the students of that program. So, this green path gives us the students that are

subscribed to a program the course is part of. Clearly, navigating along the blue path will give you another set of students than when navigating along the green path.





Let's illustrate this with a concrete Level 0 example. Here you find 3 students, 3 programs and 6 courses. The orange lines indicate the relationships between students and courses: which student has taken which course in his or her individual study program. The purple lines show the relationships between courses and programs: which course is part of what program. And the black lines tell which student is registered for what program. Let us now start from the course Object Oriented Programming. Navigating along the blue path, means navigating from the course to the students taking that course. We find Maria Haven. Navigating along the green path, means first navigating to the programs this course is part of. We find the bachelor of Computer Science. We then navigate to the students registered for this program and We find Peter Martens and Elisa Smith. So the two paths yield a different set of students.

This paper continues in UML ADVANCED