

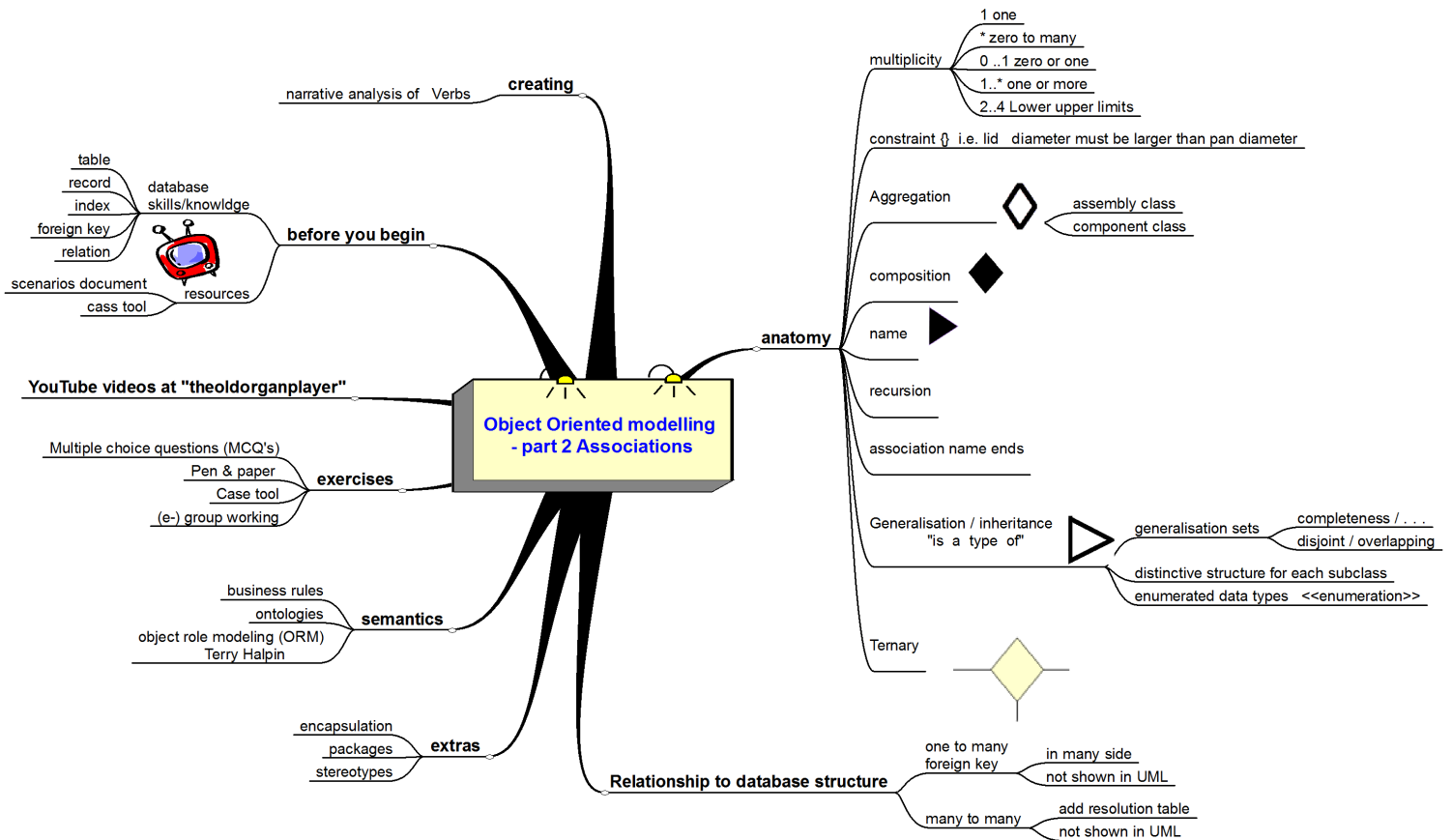
Introduction to UML (Unified Modelling Language)

Part two - Associations

Written by: Robin Beaumont e-mail: robin@organplayers.co.uk

Date last updated: Tuesday, 12 July 2011

Version: 1



You can see Youtube videos of the concepts discussed in this document at www.youtube.com/theoldorganplayer

How this document should be used:

This document has been designed to be suitable for web based and face-to-face teaching. The text has been made to be as interactive as possible with exercises, Multiple Choice Questions (MCQs) and web based exercises.

If you are using this document as part of a web-based course you are urged to use the online discussion board to discuss the issues raised in this document and share your solutions with other students.

Whom this document is aimed at:

This document is aimed at three types of people:

- Those who wish to become involved in systems development but are not interested in the nuts and bolts of programming. Such people are commonly called domain experts and act as bridges between a professional group (e.g. medics, Solicitors etc) to which they belong and IT experts.
- Those just beginning professional computer science courses.
- Those who wish to become involved in some form of analysis activity. This might be Process re-engineering or Data flow analysis etc.

I hope you enjoy working through this document.

Robin Beaumont

You can see Youtube videos of the concepts discussed in this document at www.youtube.com/theoldorganplayer

Contents

| | | |
|-----------|---|-----------|
| 1 | BEFORE YOU START | 4 |
| 2 | LEARNING OUTCOMES | 4 |
| 3 | BACKGROUND | 5 |
| 4 | BASIC UML ASSOCIATION TYPES | 5 |
| 4.1 | MULTIPLICITY..... | 6 |
| | EXERCISE 1. ASSOCIATIONS..... | 7 |
| 4.2 | AGGREGATION AND COMPOSITION \diamond \blacklozenge | 7 |
| | EXERCISE 2. AGGREGATION AND COMPOSITION..... | 8 |
| | EXERCISE 3. PRESCRIPTION ASSOCIATIONS | 9 |
| 4.3 | ASSOCIATION NAMES..... | 10 |
| 4.4 | CONSTRAINTS | 10 |
| 5 | ANALYZING A NARRATIVE DESCRIPTION TO IDENTIFY ASSOCIATIONS | 11 |
| | EXERCISE 4. USING VERBS TO HELP DEFINE ASSOCIATIONS..... | 11 |
| | EXERCISE 5. MCQS..... | 12 |
| | EXERCISE 6. ACCESSING MULTIPLICITY FROM NARRATIVES | 13 |
| 6 | COMPLEX ASSOCIATIONS | 15 |
| 6.1 | RECURSION | 15 |
| 6.2 | ASSOCIATION END NAMES | 15 |
| 6.3 | GENERALIZATION / SPECIALISATION / INHERITANCE = 'IS A TYPE OF' | 16 |
| 6.3.1 | <i>Constraining</i> | 16 |
| 6.3.2 | <i>Different structure for each subclass</i> | 17 |
| 6.3.3 | <i>Generalisation Sets</i> | 17 |
| 6.3.4 | <i>Power types - an advanced concept</i> | 18 |
| 6.3.5 | <i>Generalization - Hospital Examples</i> | 20 |
| | EXERCISE 7. GENERALISATION/INHERITANCE | 20 |
| 6.3.6 | <i>Misuse of inheritance - Enumerated data types</i> | 21 |
| | EXERCISE 8. MISUSE OF INHERITANCE..... | 21 |
| 6.3.7 | <i>Polymorphism</i> | 21 |
| 6.4 | TERNARY ASSOCIATIONS..... | 22 |
| 6.4.1 | <i>Breaking Ternary associations into Binary ones</i> | 23 |
| 7 | MAPPING ASSOCIATIONS TO DATABASES | 25 |
| 7.1 | ONE TO MANY RELATIONSHIPS..... | 25 |
| | EXERCISE 9. CONVERTING CLASSES/INSTANCES TO TABLES/RECORDS | 25 |
| 7.2 | MANY TO MANY RELATIONSHIPS | 26 |
| | EXERCISE 10. MCQS..... | 26 |
| 8 | THE MEANING OF ASSOCIATIONS (SEMANTICS) AND BUSINESS RULES | 27 |
| 9 | ENCAPSULATION | 28 |
| 9.1 | PACKAGES | 28 |
| 9.2 | STEREOTYPES | 29 |
| 10 | SUMMARY | 30 |
| 11 | REFERENCES | 31 |
| 12 | APPENDIX - OMT | 32 |

1 Before you start

This document assumes that you have worked through the first chapter - Introduction to UML - Part one - Classes and instances available at: <http://www.robin-beaumont.co.uk/virtualclassroom/contents.html> This chapter contains a list of learning outcomes including specific **Skills/ knowledge** you will have gained working through the chapter.

In writing this chapter I have assumed that you have the skills/knowledge developed in this previous chapter

Required Resources

You need the following resources to work through this document:

- The “Scenarios for practicing modelling techniques” handout available from <http://www.robin-beaumont.co.uk/virtualclassroom/contents.html> and follow the links
- A UML case tool - Many of the concepts introduced in this document are difficult to grasp at first and are helped by experimenting with a Case Tool in addition to carrying out the exercises with pen and paper. Two such tools are Visual UML and MagicDraw. You can download a community version of MagicDraw at <http://www.magicdraw.com/>. In addition if you are reading this document as part of a course you are undertaking with the Royal college of Surgeons (Edin) or Edinburgh University you are registered as part of the academic programme for Magicdraw which means you are entitled to use the full personal edition of MagicDraw. To be able to use the full personal edition you need to contact your course administrator who has the codes to unlock MagicDraw.

2 Learning Outcomes

This chapter aims to provide you with the following skills and knowledge. After you have completed it you should come back to these points, ticking off those with which you feel happy.

| Learning outcome | Tick box |
|--|--------------------------|
| Be able to identify and draw the different types of associations between classes including end names | <input type="checkbox"/> |
| Be able to describe the concepts of aggregation and composition | <input type="checkbox"/> |
| Be able to describe the way association lines in class models are implemented in a database | <input type="checkbox"/> |
| Be able to describe briefly the concept of semantic modelling and business rules | <input type="checkbox"/> |
| Be able to describe, and provide an example of generalisation/inheritance | <input type="checkbox"/> |
| Be able to describe the various ways of modelling generalisation/inheritance | <input type="checkbox"/> |
| Be able to demonstrate the use of enumerated data types in place of inappropriate generalisation/inheritance | <input type="checkbox"/> |
| Be able to demonstrate the usefulness of generalisation/inheritance when modelling healthcare concepts | <input type="checkbox"/> |
| Be able to describe the concept of polymorphism | <input type="checkbox"/> |
| Be able to develop a class model for a familiar area | <input type="checkbox"/> |

3 Background

So far we have considered classes as being distinct discrete concepts with no linkage between them, however in reality classes obviously link to one another in many different ways and UML provides methods for modelling this.

4 Basic UML association types

When modelling a system, in terms of classes it is vital to be able to define the way classes relate to each other. This is achieved by considering **associations** between classes. Although association modelling can be at a number of levels of complexity, we will only consider some of the simpler techniques available.

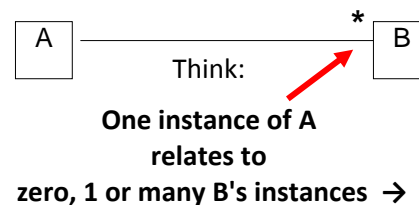
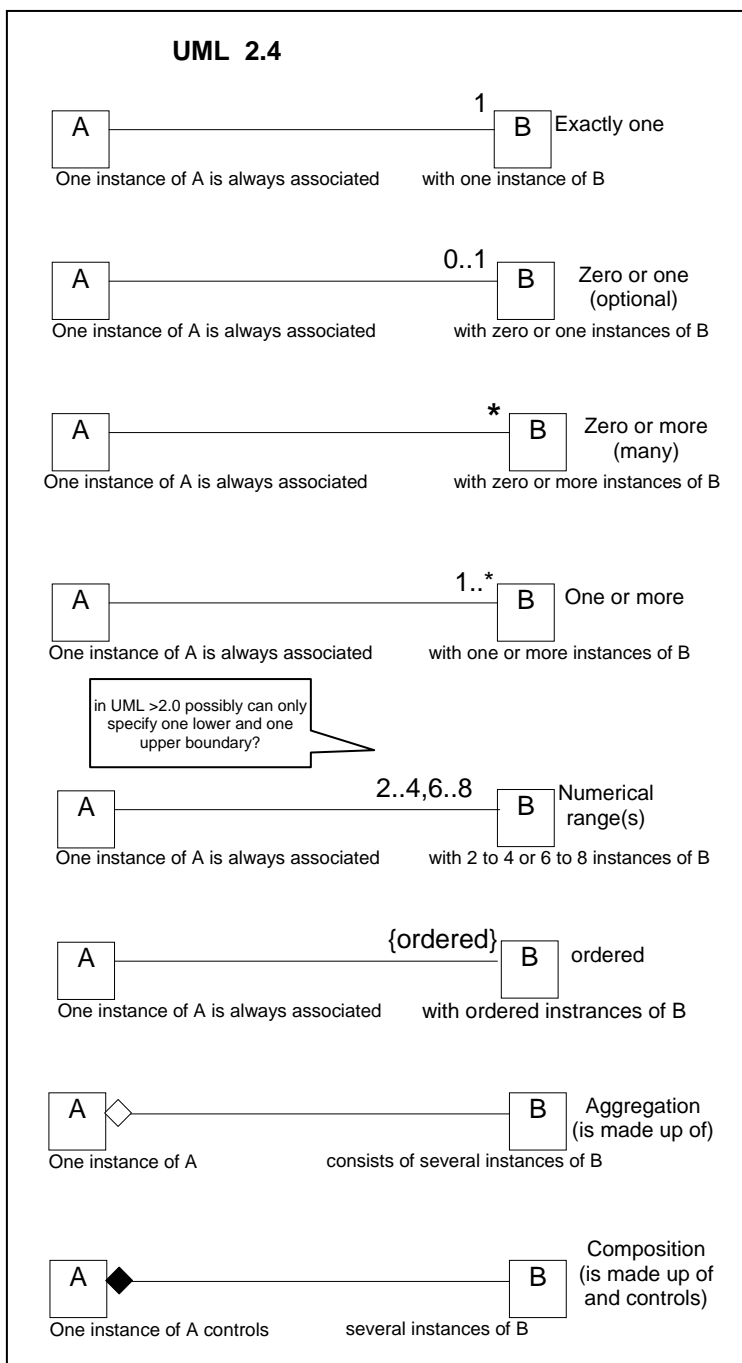
An association often appears as a verb in a sentence, such as:

A person [class] *eats* bananas [class] or Sally [instance] *eats* the banana [instance] on her desk

An association at the instance level is called a **link**.

Once again UML has a sets of symbols to describe what are considered to be the common types of associations that exist between classes I have listed most of them opposite for easy reference:

The multiplicity indicator, that is the 1, or 0..* or * etc. refers to the class close to it (i.e. proximal for the medics among you) on the line



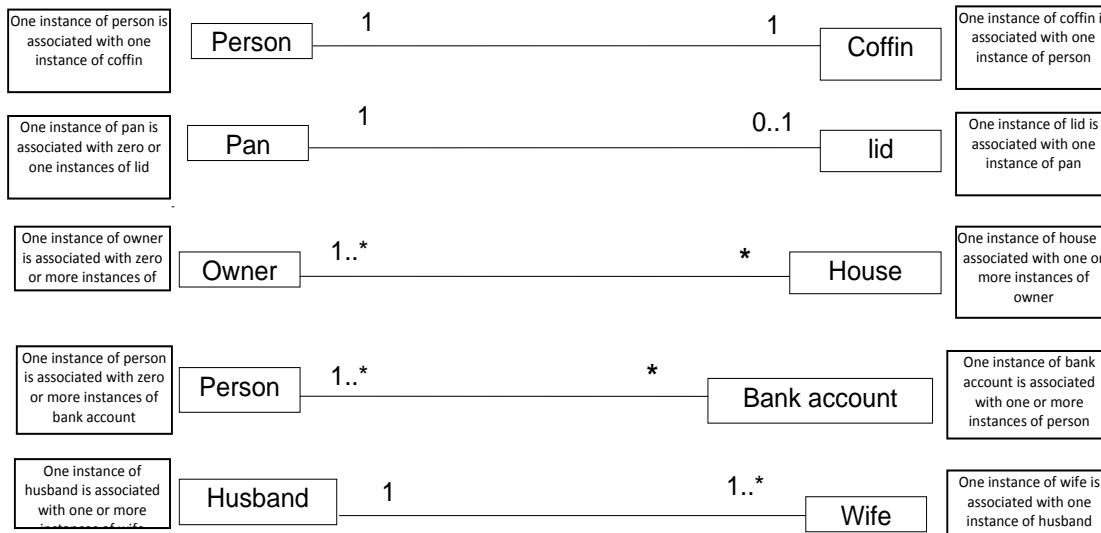
All of the associations in the diagram opposite provide information about how many instances of one class can be associated with ('related to') another. Because these associations are concerned with the number of occurrences, they are often classified by their **'multiplicity'** and on the next page examples are given of each type of multiplicity.

It should be noted that working out the type of relationship that exists between classes is often done after it is agreed that some type of relationship exists. Therefore a first draft of a class diagram may possess many unmarked ('unspecified') associations which are only subsequently elucidated.

We will begin by looking at the first four associations.

4.1 Multiplicity

Multiplicity specifies how many instances of one class relate to a single instance of an associated class. The simplest type of multiplicity is that of one-to-one. This is shown as the first example below. Two classes that have a one-to-one association will therefore be drawn with a simple line between them with '1' at each end.. Such an example might be a needle and syringe or a person and coffin. Considering a slightly more complex situation, a frying pan may optionally have a lid, which would be represented by using the "0..1" multiplicity expression. You will also notice here that I have multiplicities at both ends of the association line.



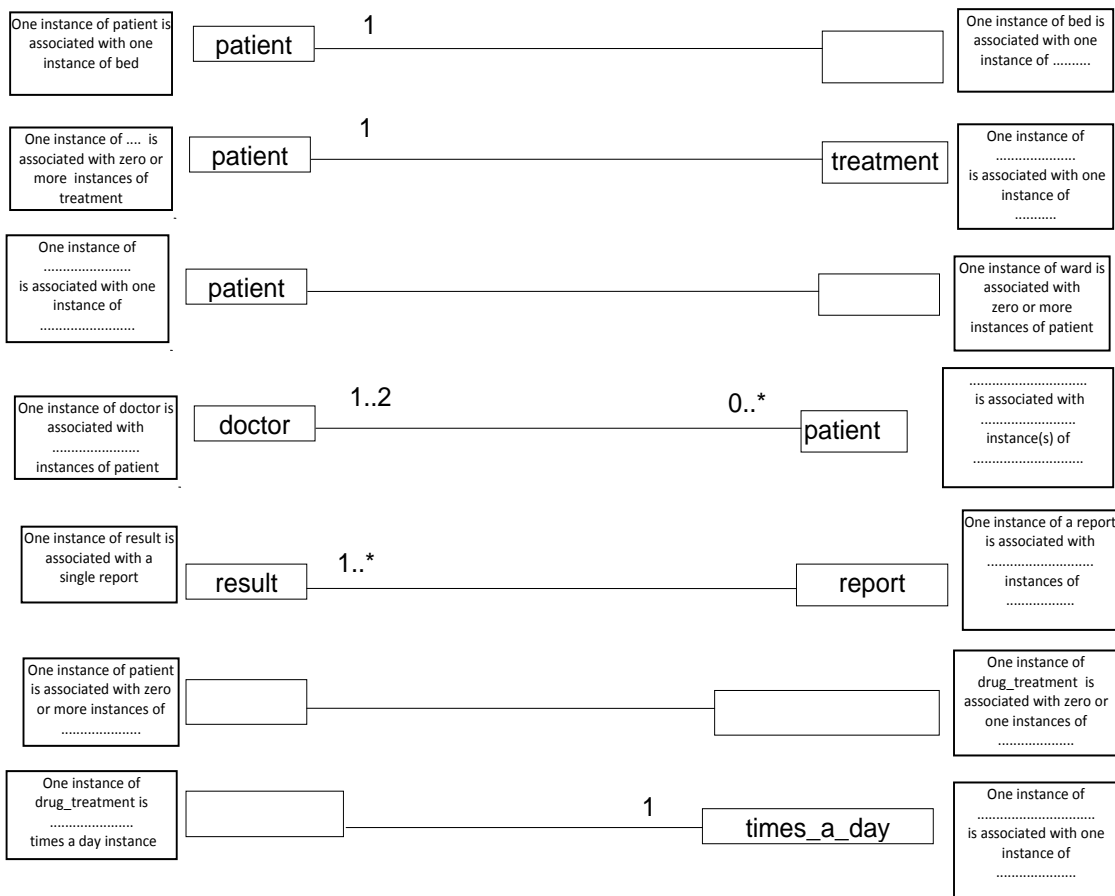
If the modeller wishes to explicitly state how many occurrences of one class are associated with another, a value can be placed at the relevant end of the association line. For example, if a team leader always has four or more case workers this situation would be represented with '4+' above the association line. Discrete values can also be modelled such as in the situation where a team leader can only have 3, 5 or 6 case workers; this situation would be represented by '3,5,6' above the association line in versions of UML before version 2 there seems to be no mention of the use of such discontinuous sets which have again disappeared in version 2.4.

Multiplicity is directional; the end of the line where the symbol occurs relates to the class at that particular end of the line. All the symbols mentioned above can occur at either end of the association. A few examples are given above using the UML notation.

If you know about ERD's and the crows feet notation you will have realised that the above types of associations are basically the same as those. However UML provides a much greater variety of association types not found in ERDs and we will now move onto one of these, that is the diamond shape which we will discuss next.

Exercise 1. Associations

Complete the following association by either completing the missing sentences, adding the correct cardinality symbols or adding the appropriate class names.

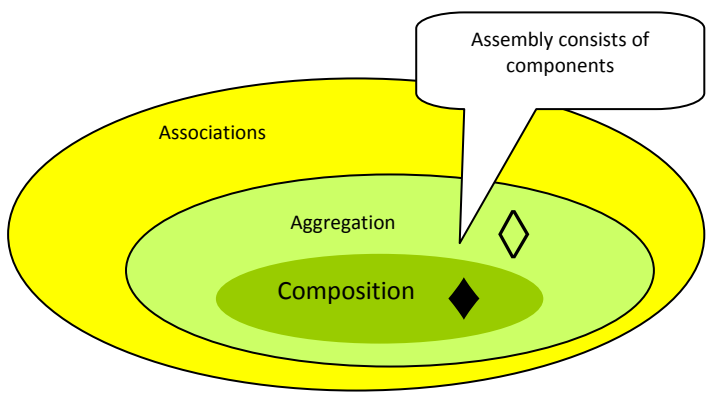


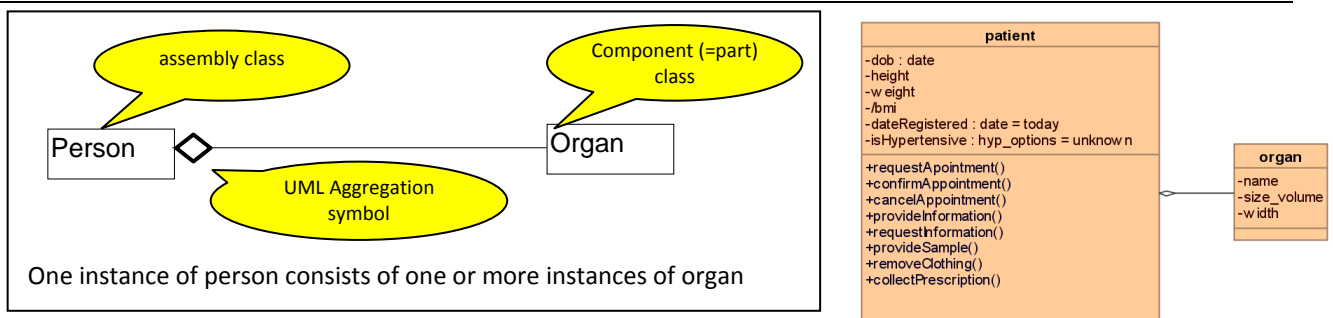
Having now considered the simplest type of UML associations we will now move onto some of the more complex types.

4.2 Aggregation and Composition

As would be expected from UML these are more specialised types of association. Let's consider aggregation first.

Aggregation (technically called *aggregationKind = shared* UML superstructure specification v2.4 p. 24) can be translated into words as meaning **consists of**. Lets consider two classes; PERSON and ORGAN, we can say that a person consisted of organs. In technical speech we would say that the person was the '**assembly**' class and the organ was the '**component**' or **part** class. The assembly class can be thought of as a type of collection. Taking another example, a degree consists of a collection of units/modules.





Obviously the above can be 'fleshed' out by considering the PATIENT class we developed in the previous chapter as shown in the above diagram on the right and from this you can see that the assembly class may add additional attributes/operations that the sum of the component classes may not possess. so while each individual ORGAN has a name the PATIENT adds additional details when they are all combined such as BMI etc.

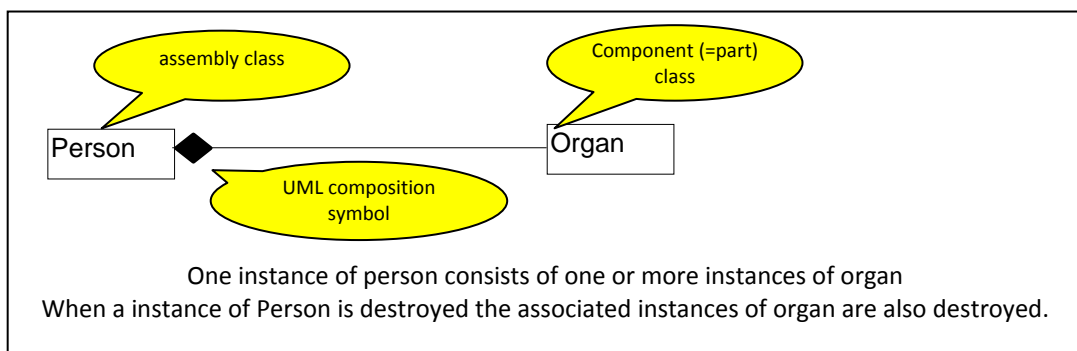
An assembly class ('collection') can be made up of more than one type of component. For example, a hospital hopefully consists of collections of both patients and staff.

Fowler, 2004 p.67 considered aggregation to be a problem because different people interpret it differently; he quotes Jim Rumbaugh in saying "think of it as a modelling placebo". I thought that the medics among you would appreciate that! This view is reinforced by the UML superstructure specification v2.4 p. 39 "The Precise semantics of shared aggregation varies by application area and modeler. The order and way in which part instances are created is not defined."

Composition (technically called *aggregationKind = composite* UML superstructure specification v2.4 p. 24) is more clearly defined as it includes the constraint of co-existence in effect the assembly classes controls the existence of the component instances. In other words in composition the assembly class is in charge of destroying the component class and this implies that both parts have a degree of co-existence the part can not exist without the 'whole'.

Thinking about the above definitions of Aggregation and Composition you can see why it is often difficult to decide which to use. For example, take the situation of the PERSON has ORGANS. To a priest this is a clear example of Composition; an ORGAN can only exist as part of a PERSON. To an organ transplant co-ordinator, however, it is very much an aggregation; organs certainly do exist on their own! Again, this demonstrates how models are context specific.

Graphically the difference between aggregation and composition is that aggregation is an open diamond while composition is filled.

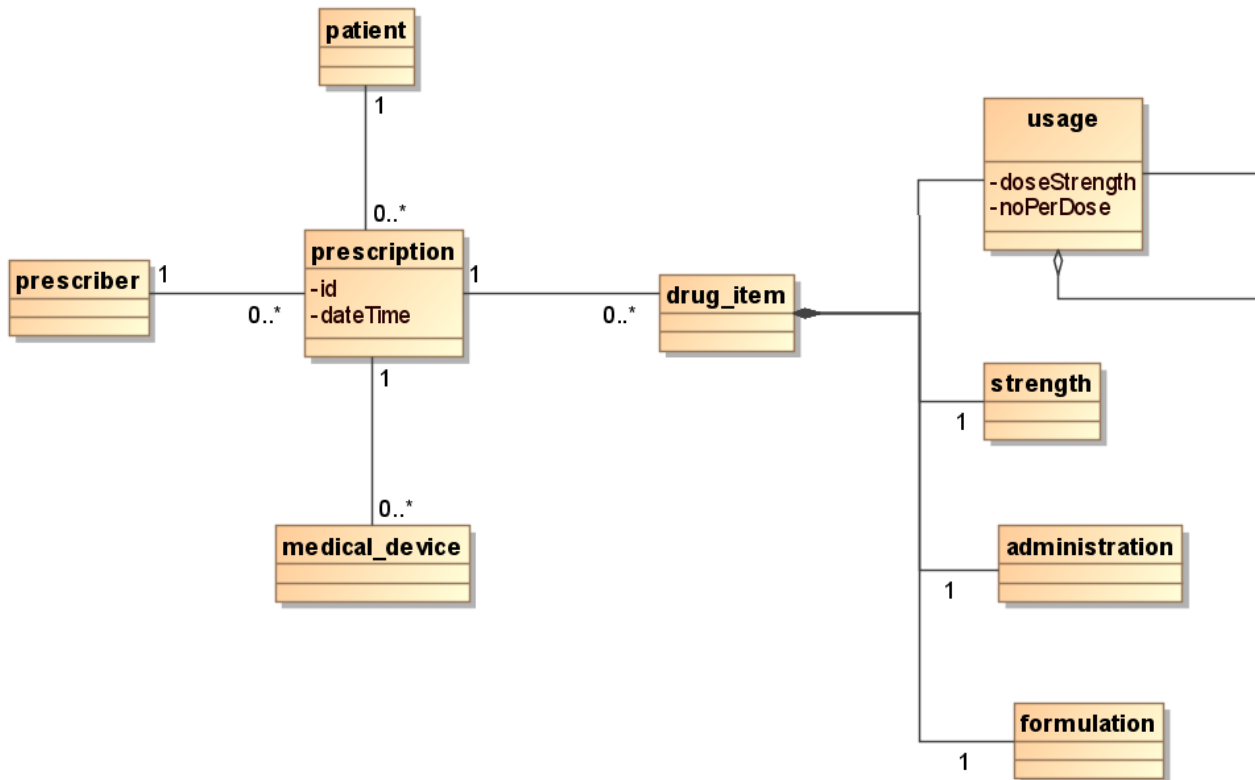


Exercise 2. Aggregation and Composition


Draw class diagrams showing the data for each of the following aggregations / compositions: bed, ward and hospital, canteen department and a department of your own choice.


Exercise 3. Prescription associations

The following class diagram is of a prescription. Formulation is to do with how the drug is prepared (e.g. tablet, paste etc). Administration route is the method of taking the drug (e.g. oral etc), and usage details are basically instructions for the patient besides administration route.



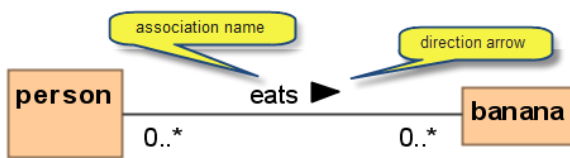
1. In the above diagram, which is the **incorrect** statement?
 - a. An instance of prescription has zero or more drug items.
 - b. An instance of medical device is related to a single prescription.
 - c. Each instance of drug item consists of usage, strength, administration (route) and formulation details.
 - d. An instance of drug item will contain a single instance of usage.
 - e. A single drug item can be related to a single prescription.

2. In the above diagram, what does the symbol  indicate?
 - a. Usage details consist of one or more such sets of details.
 - b. Usage details consist of zero or more such sets of details.
 - c. Usage details are related to several other usage details.
 - d. Usage details are not clearly defined.
 - e. The relationship is ambiguous.

3. In the above diagram, what does the  symbol along with the multiplicity values indicate:
 - a. A drug item must contain details of one or more of the following: usage, strength, administration and formulation details.
 - b. A drug item must contain details of one instance of strength, administration and formulation details but possibly more than one set of usage details.
 - c. A drug item is only 'complete' when it contains information about strength, administration and formulation.
 - d. A drug item is only 'complete' when it contains information of usage, strength, administration and formulation.
 - e. A instance of formulation is associated with a single drug_item.

4.3 Association names

The verb is frequently used to provide an **association name** for the association. If this is the case it is placed somewhere beside the association line on the diagram with an optional arrow beside it. [See Kirill Fakhroutdinov's UML website discussion of the ambiguity in the UML specification concerning the direction arrow at <http://www.uml-diagrams.org/uml-core.html#core-association>]



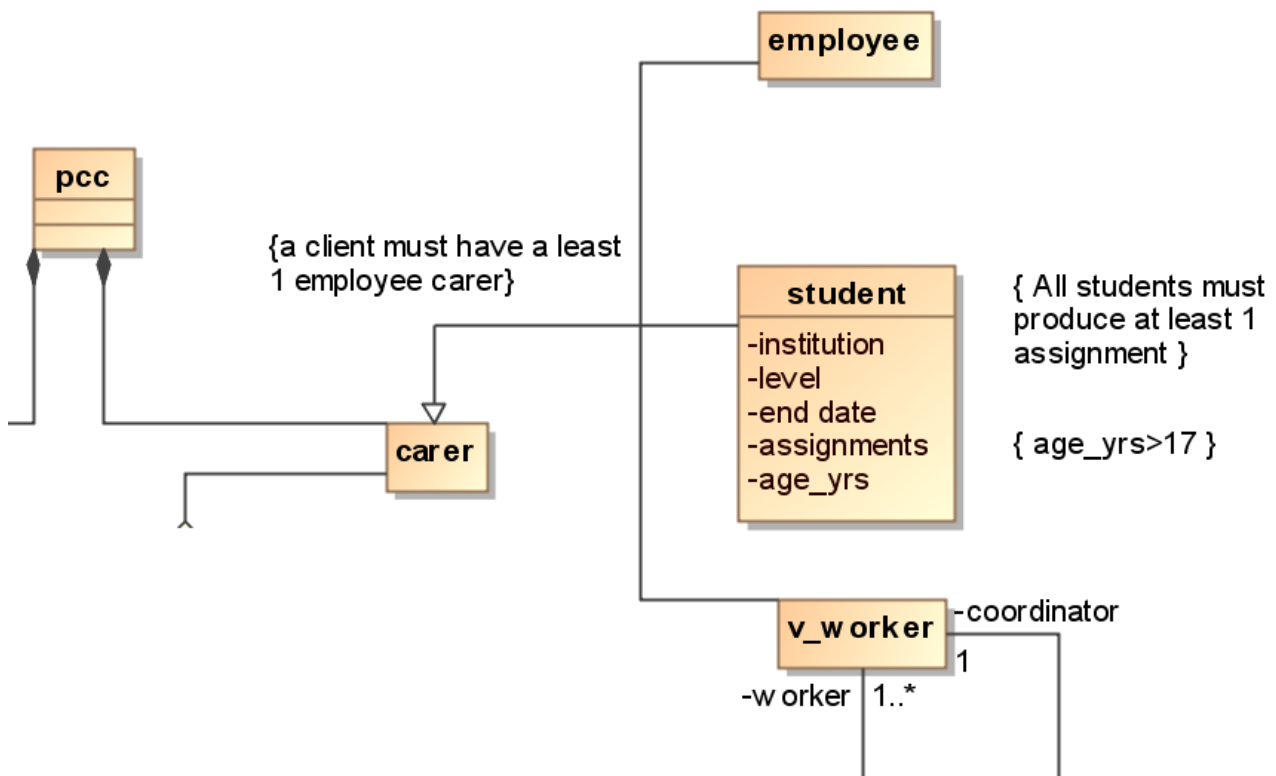
A person *eats* zero, 1 or more bananas, and a banana is eaten by zero, 1 or more people:

The bottom line:

I don't use association names often as I feel they add unnecessary clutter to diagrams - it is largely a matter of personal taste.

4.4 Constraints

UML constraints are used in many places, they are usually short sentences which can be Boolean expression, that is can be evaluated as true/false. The only requirement is that the expression is enclosed in curly braces {}. The example below shows three constraints. One concerning the number of instances associated with the CLIENT class (not shown) and two concerning attributes of the STUDENT class. UML has a special language OCL (Object Constraint Language) that uml gurus use to write the expressions. Bennett, Skelton & Lunn 2005 provide a whole chapter describing it.



Often it is best to put complex constraints in the accompanying text to the diagram, for example I personally would not have put the three above constraints on the diagram.

5 Analyzing a Narrative Description to identify Associations

Referring back to the narrative used in developing classes in the previous chapter we now focus on the **verbs** in the text:

We wish to develop an information system for our hospital in which:

*“Patients are **treated** in a single ward by the doctors **assigned** to them. Usually each patient will be **assigned** a single doctor, but in rare cases they will have two. Patients either **pay** for their treatment directly or through an insurance company.*

*Healthcare assistants also **attend** to the patients, and a number of these are **associated with** each ward.*

*Initially the system will be concerned solely with drug treatment. Each patient is required **to take** a variety of drugs a certain number of times per day and for varying lengths of time.*

*The system must record details concerning patient treatment and staff payment. Some staff are **paid** part time, and doctors and care assistants **work** varying amounts of **overtime** at varying rates (subject to grade).*

*The system will also need to **track** what treatments are required for which patients and when, and it should be capable of **calculating** the cost of treatment per week for each patient.*

*When users use the system they will be able to **print out** as well as **view** on screen the results. (Although it is currently unclear to what use this information will be put.) “*

(taken from http://www.umsl.edu/~sauter/analysis/er/er_intro.html and expanded)

The strategy

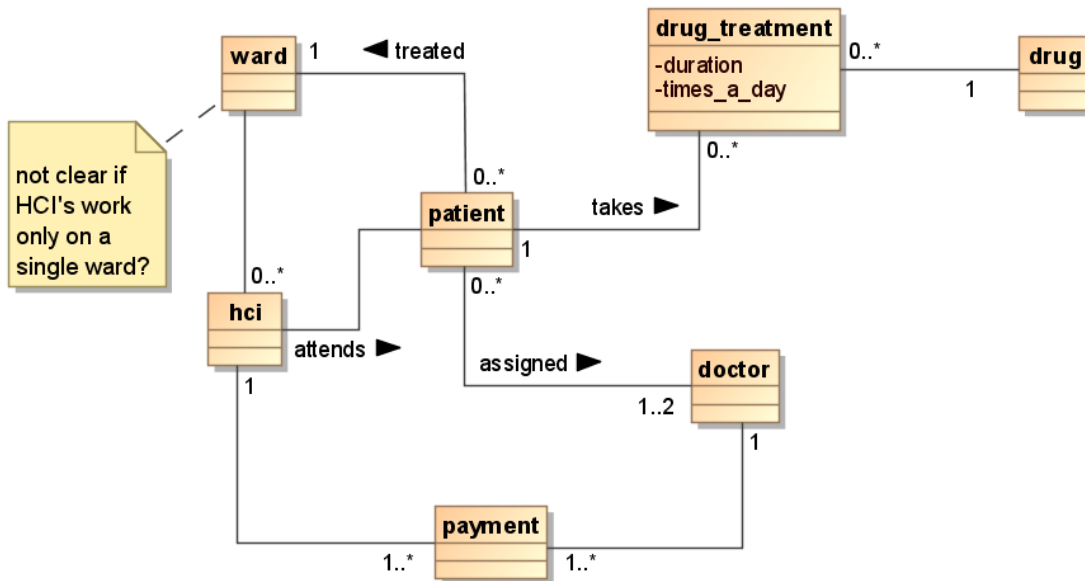
The above edited narrative suggests many associations one group of associations that should be ignored at the design stage is that concerned with **implementation** for example calculating, print out and view. Similarly results can be thought of a type of report basically this is achieved by the software system interrogating the various classes and then producing a summary - this should be ignored at the design stage of modelling.

Exercise 4. Using verbs to help define associations

Considering the classes developed in the previous chapter (PATIENT, DOCTOR, DRUG_TREATMENT, DRUG) along with the highlighted verbs above create some associations

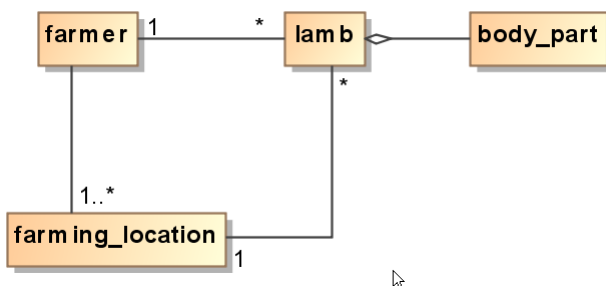
You may need to create some additional classes.

You can see below the model that I have come up with, this is not a definitive answer but one of several possibilities.



Exercise 5. MCQs

The following questions relate to the class diagram below, concerning farmers, farming locations and lambs.



1. In the above diagram, which is the correct statement?

- A farmer has one or more lambs.
- A farmer has one lamb.
- A farmer has zero or more lambs.
- A lamb does not necessarily have a farmer associated with it.
- It is impossible to ascertain the relationship between farmers and lambs in the above diagram.

2. In the above diagram, which is the incorrect statement?

- A farmer has one or more farming locations.
- A farming location is associated with zero or more lambs.
- A lamb is always associated with one farmer.
- A lamb can be associated with more than one farming location.
- A lamb consists of body parts.

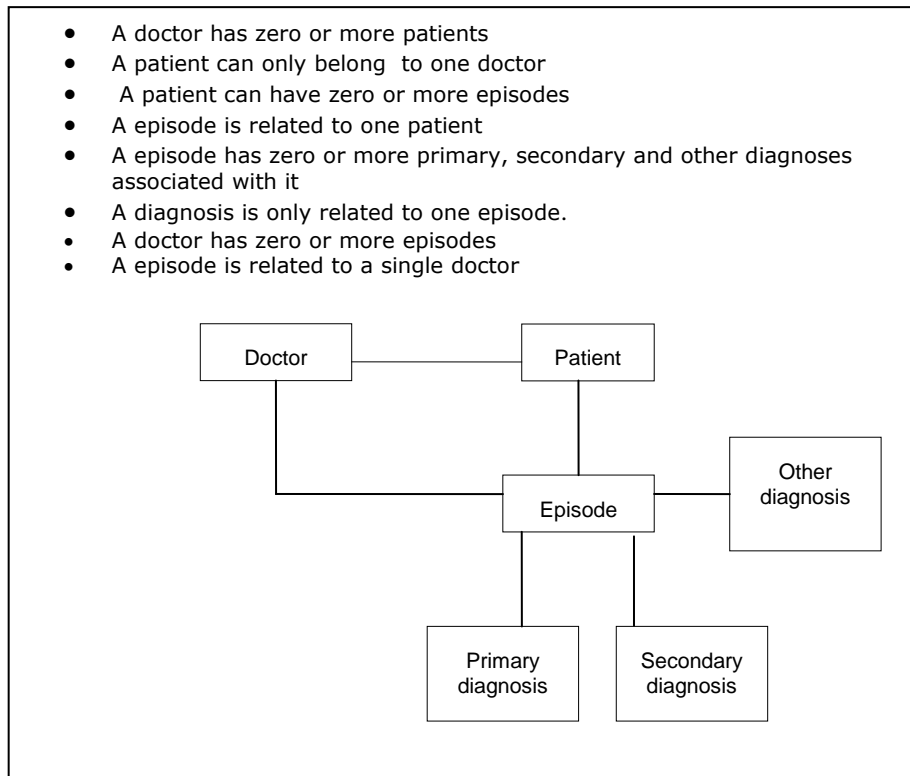
3. The above diagram uses:

- OMT notation
- UML notation
- ERD notation
- No formal notation
- Chen notation

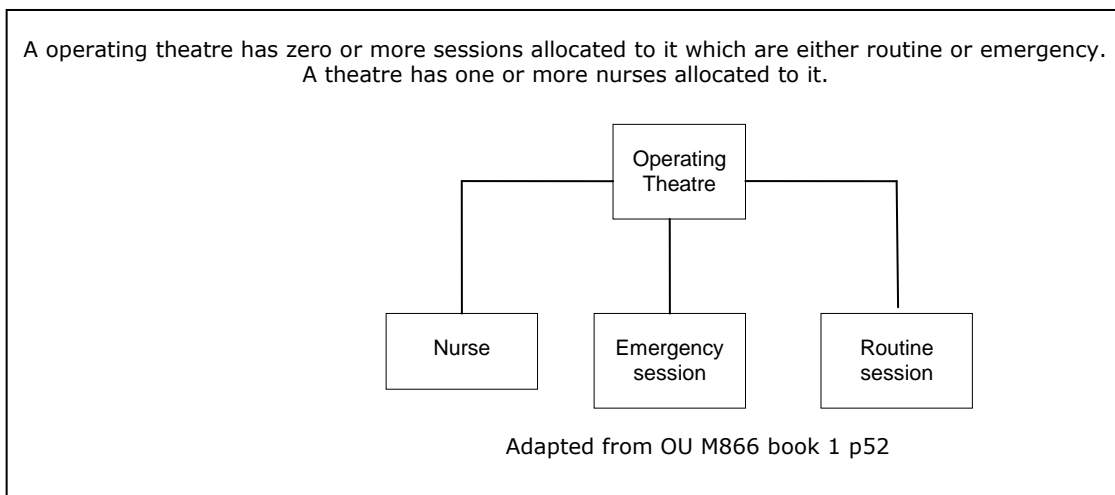
Exercise 6. Accessing multiplicity from narratives

This exercise requires you to complete the following models adding the appropriate UML multiplicity symbols and if you wish association names.

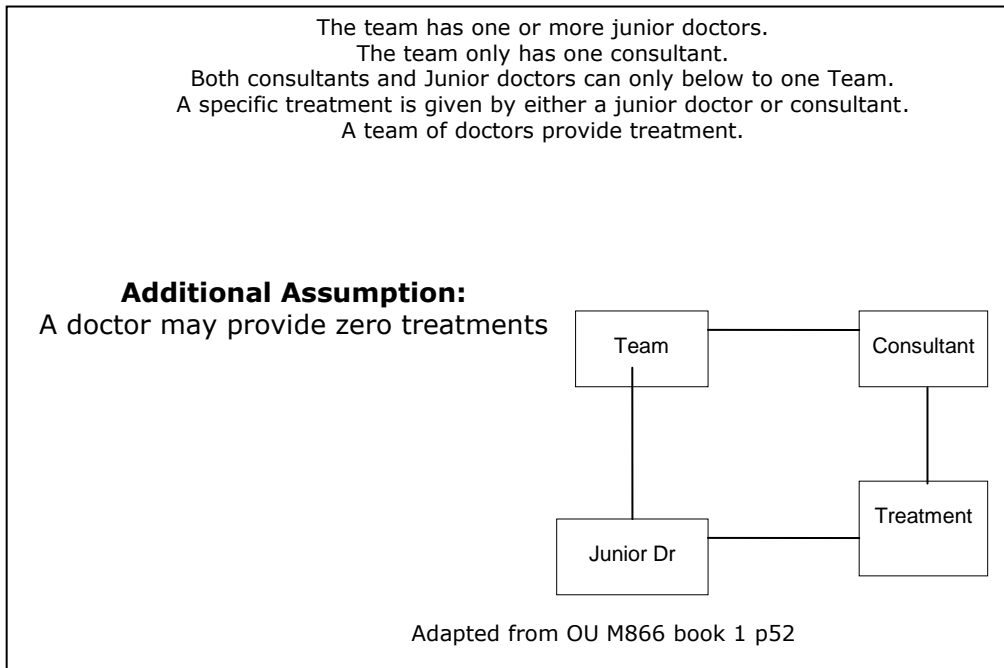
1. General Practitioner (GP) consultation model:



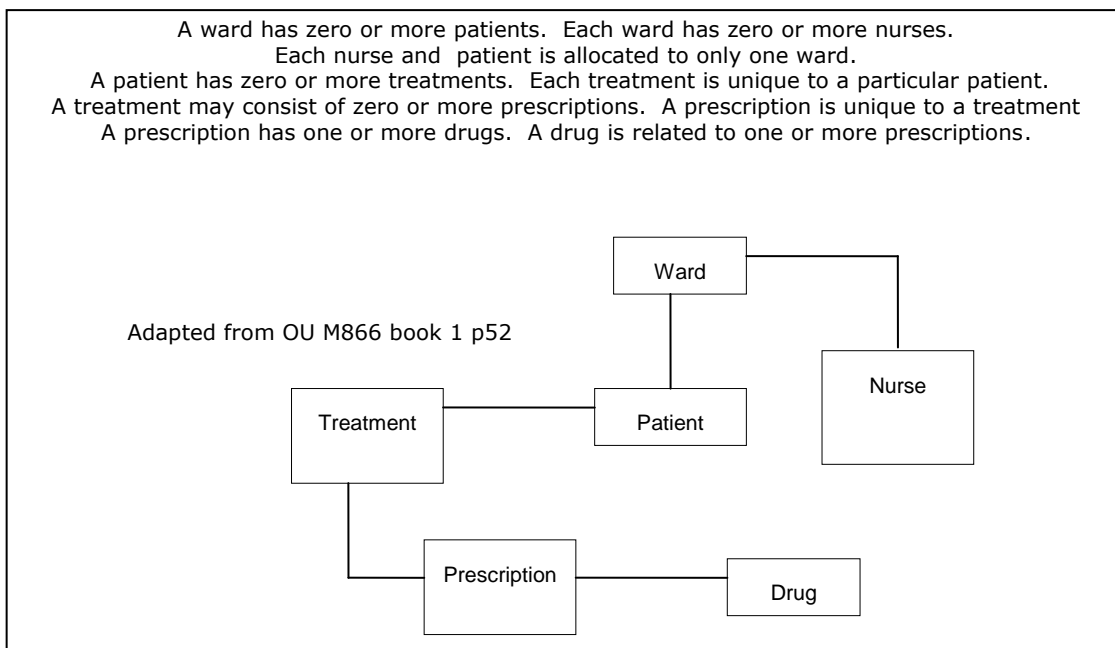
2. The hospital model part A:



3. The hospital conceptual model part B (adapted from Open University M866 book 1 p52):



4. The hospital model part C (adapted from OU M866 book 1 p52):



6 Complex associations

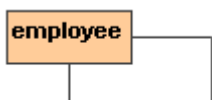
In this section we will consider ways of modelling more complex associations including:

- Recursive
- Association end names
- Inheritance/Generalisation sets
- Ternary Associations

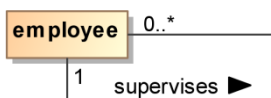
6.1 Recursion

Recursive associations are also called **unary** or **involved** associations.

So far we have discussed examples of associations between classes; however, it is possible to have an association between instances in the same class. Such a association is called recursive.



The usual example given is that of an EMPLOYEE class that has a relationship with itself called supervisor. In other words an EMPLOYEE instance can relate to another EMPLOYEE instance in a supervisor role. The recursive association opposite has an undefined multiplicity, however clearly a 'one to many' or 'many to many' recursive association may exist. (see Carter 1995 p61 - 68, Elmasri & Navathe 1989 p. 49). Some writers suggest that these recursive associations are acceptable in early models but should be converted to 'one to many' or 'one to one' associations, as the model is refined by creating a new class. For example the above recursive association could be modelled by introducing a new class type called ROLE. Some modellers argue that it is best to abstract out these recursive associations, particularly if the final model is going to be implemented in a database.



It is easy to see how this is done if you consider the above example, giving the recursive association a one to many association (opposite) and using a non UML table.

The table below shows 6 instances of employee with values for both the ID and supervisor_id attributes, where the supervisor_id attributes refers to the values in the ID field. So instances 1, 3, 5 and 6 are supervised by employee instance 4, similarly instances 2 and 4 are supervised by employee instance ID 5.

| Class / instances. Class name: employee | | | | | |
|---|---------------|-----|-----|-------|-----|
| ID | Supervisor_id | ... | ... | | etc |
| 1 | 4 | | | | ... |
| 2 | 5 | | | | ... |
| 3 | 4 | | | | ... |
| 4 | 5 | | | | ... |
| 5 | 4 | | | | ... |
| 6 | 4 | | | | ... |

Removing the SUPERVISOR_ID attribute and copying the ID attribute into a new class SUPERVISOR results in the following situation given below.

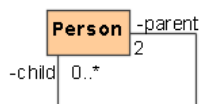


| Class / instances. Class name: employee | | | | | |
|---|-----|-----|-------|-----|--|
| ID | ... | ... | | etc | |
| 1 | | | | ... | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |

| Class / instances. Class name: supervisor | | | | | |
|---|---------------|-----|-----|-------|-----|
| ID | Supervisor_id | ... | ... | | etc |
| 1 | 4 | | | | ... |
| 2 | 5 | | | | ... |
| 3 | 4 | | | | ... |
| 4 | 5 | | | | ... |
| 5 | 4 | | | | ... |
| 6 | 4 | | | | ... |



Incorrect model

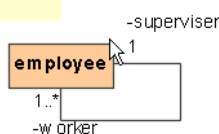


correct model

In contrast to moving out the recursive associations and creating 'Role' classes Blaha & Rumbaugh 2005 p. 32 suggests that recursive associations should be retained using association end names as shown in the example opposite. Now lets see what these association end names are actually.

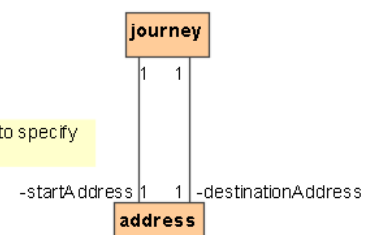
6.2 Association End Names

end name to specify role



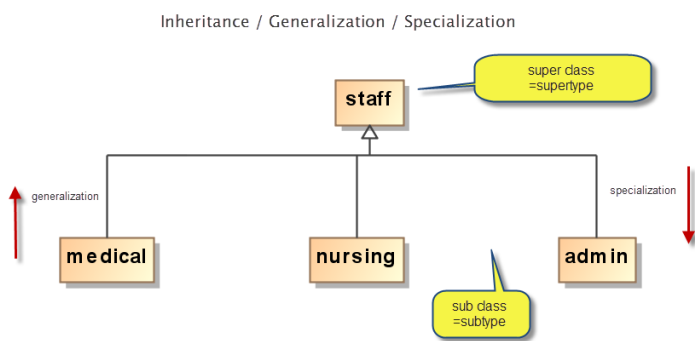
In UML the concept of Association end names are used to specify a role a particular instance may play in the association. Considering the above example an EMPLOYEE may only have one supervisor and the supervisor will manage one or more employees.

end name to specify role



Association end names may also be used to clarify the situation where several associations exist between a single class. For example consider the situation where a courier makes journeys which require both a start address and a destination address, this can easily be modelled using association end names as shown in the diagram opposite (adapted from p. 97 Bennett, Skelton & Lunn 2005).

6.3 Generalization / Specialisation / Inheritance = 'is a type of'



All three of the above terms relate to the same process of creating several classes that relate to one another in a specific way, this is a very powerful tool for organising what often appears to be a very complex situation into one that is more manageable. When modelling, this is one of the later actions that takes place, that is after all the classes and associations have been quite thoroughly described and, as usual involves the development of several drafts.

Inheritance (the open triangle symbol in UML) represents the process of inheriting data and/or operations from a parent (**superclass**) to that of a child class (**subclass**). An example would be the subclass MEDICAL (staff) who would inherit all the detail in the STAFF superclass but

additionally add some of their own. Often the parent 'super class' is something that exists in the model but not in the real world, you can think of it as bag of commonalities used by all the subclasses. In our STAFF example above it is unlikely that instances of STAFF exist only its subtypes. When no instances of the parent class exist, it is known as an **abstract class**.

Generalisation is just describing inheritance from the super class perspective as the super class is a generalization of the various subclasses. In contrast the subclass is a **specialization** of the superclass.

In a UML Class diagram you show inheritance by using either a single triangle, or one for each line, but this can lead to confusion which we will see why in a subsequent subsection. The association line when it has a generalisation symbol at the end of it is called a **generalisation line**.

6.3.1 Constraining

A constraint is a UML defined term that is shown in curly brackets { . . . } near the inheritance triangle on the diagram. In this situation the constraint consists of two words, each representing one of two mutually exclusive options.

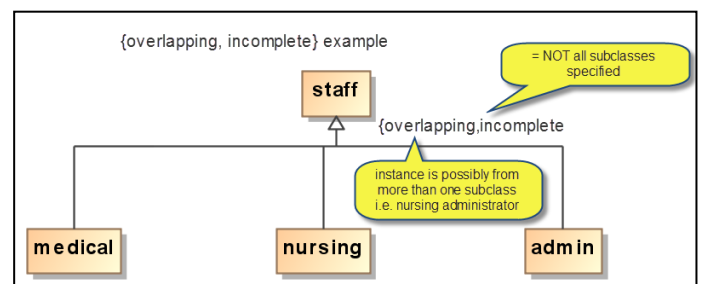
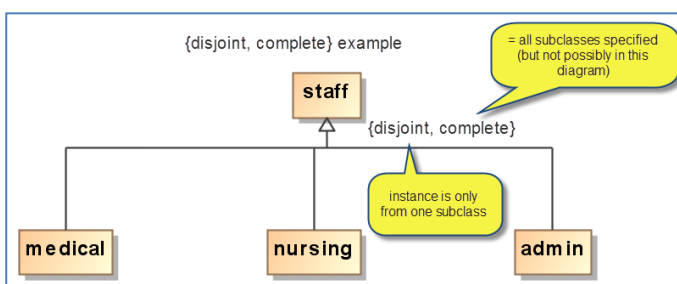
First mutually exclusive option:

- **Complete** – All children (i.e. subclasses) have been specified (whether or shown not). No additional children are expected.
- **Incomplete** – Some children (i.e. subclasses) have been specified, but the list is known to be incomplete. There are additional children that are not yet in the model. This is a statement about the model itself. Note that this is not the same as the ellipsis, which states that additional children exist in the model but are not shown on the current diagram.

Second mutually exclusive option:

- **Disjoint** – An instance can only be from one of the subclasses within the generalisation set (i.e. either medical, nursing or admin).
- **Overlapping** – An instance can be from one or more of the subclasses within the generalisation set (i.e. possibly a combination of medical, nursing or admin)..

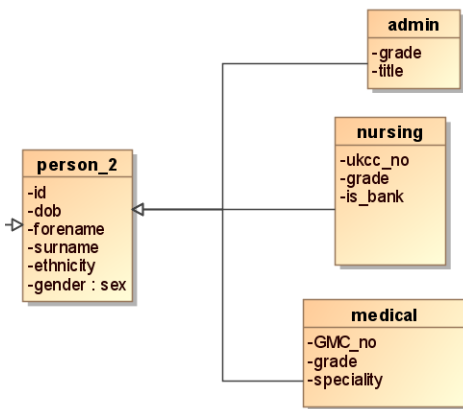
If no constraint is shown on the generalisation line, the default is **{incomplete, disjoint}** which is by far the most common of the four possible options two if which are shown below.



Additional subclasses not shown on a diagram because of space constraints can be indicated by three black circles • • •

• at the end of the association line. Rumbaugh (2005) also suggests you can just add another sub class called 'other' as well possibly instead of using the incomplete constraint.

6.3.2 Different structure for each subclass



When considering the subclasses it is important to realise that while you are looking for commonalities between them in terms of structure (i.e. attributes + operations) you are also aiming for them to be structurally distinct, for example the three subclasses of PERSON_2" opposite have different attributes besides the core ones they inherit from PERSON_2. They certainly would have different operations.

Also notice that they all have an attribute called GRADE, although this is the same name for each subclass you will appreciate that they will be different, as the medical grades are different to nursing grades which again are different from administrative ones. Possibly it would have been more sensible to give the attribute a different name in each subclass.

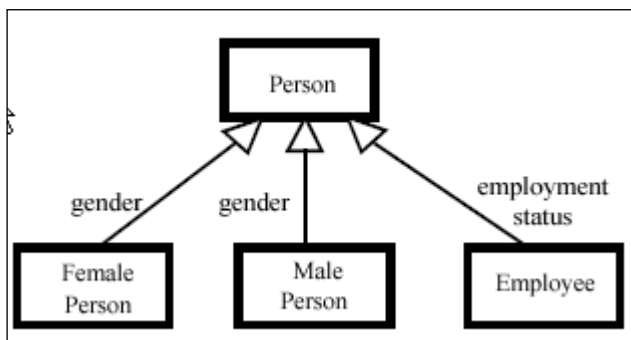
When both a superclass and a subclass have an attribute/operation with the same name what it means in effect is that the superclass definition is overwritten by the subclass one. Lets now consider more the relation between the various subclasses.

The bottom line:

Each subclass should have a different structure (attributes+ operations).

We will consider this again when looking at enumerated data types in a subsequent subsection.

6.3.3 Generalisation Sets



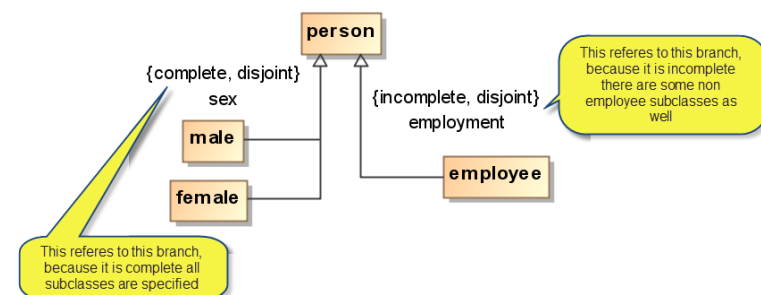
So far we have considered and/or situations for a number of subclasses however we may want to further divide the subclasses into subsets (one which might be 'or'=disjoint and another which might be and=overlapping etc) and each such subset of subclasses is called a **generalisation set**. The uml superstructure document explains this and I have copied the example below.

Does the diagram opposite mean that a employee can only be a male or female or employee? If we use the disjoint constraint then they can only be one of the three subclasses, similarly if we use the overlapping constraint

than they could be a Female+male+employee, neither makes sense, what we want to show is that the subclass is either male + employee or female + employee. Luckily this can be achieved by using **Generalisation sets** to redraw the diagram. Each generalisation set is given a name (i.e. gender and employment status).

Quoting directly from the superstructure specification 2.4 page 81:

"In the illustration below, the Person class can be specialized as either a Female Person or a Male Person. Because this GeneralizationSet is partitioned (i.e., is constrained to be complete and disjoint), each instance of Person must *either* be a Female Person or a Male Person; that is, it must be one or the other and not both. (Therefore, Person is an abstract class because a Person instance may not exist without being either a Female Person or a Male Person.) Furthermore, a Person instance can be specialized as an Employee. The generalization set here is expressed as {incomplete, disjoint}, which means that instances of Persons can be a subset as Employees or some other unnamed collection that consists of all non-Employee instances. In other words, Persons can *either* be an Employee or in the complement of Employee, and not both. Taken together, the diagram indicates that a Person may be 1) either a Male Person or Female Person, **and** 2) an Employee or not."



I have highlighted the **and** in the above sentence as while the constraints refer to the individual specialization sets, it seems that the relationship between the subsets (i.e. sex and employment here) is **AND**. That is the subclass we create takes something from each generalization set in contrast to the OR situation where it would be mutually exclusive. I can't find in the uml documentation a description of how multiple

generalisation sets relate to one another, therefore I advise you to either assume the AND situation or add notes to either the diagram or the accompanying text to clarify how you want it to be interpreted. As with ordinary associations the AND situation is the default.

6.3.4 Powertypes - an advanced concept

This is the most complex aspect of generalisation sets and I would suggest that you ignore this subsection when first reading through this chapter I have again resorted to using the UML superstructure specification v2.4 p82-3 here.

----- start of extract -----

Grouping the objects in our world by categories, or classes, is an important technique for organizations. For instance, one of the ways botanists organize trees is by species. In this way, each tree we see can be classified as an American elm, sugar maple, apricot, saguaro—or some other species of tree. The class diagram below expresses that each Tree Species classifies zero or more instances of Tree, and each Tree is classified as exactly one Tree Species. For example, one of the instances of Tree could be the tree in your front yard, the tree in your neighbor’s backyard, or trees at your local nursery. Instances of Tree Species, such as sugar maple and apricot. Furthermore, this figure indicates the relationships that exist between these two sets of objects. For instance, the tree in your front yard might be classified as a sugar maple, your neighbor’s tree as an apricot, and so on. This class diagram expresses that each Tree Species classifies zero or more instances of Tree, and each Tree is classified as exactly one Tree Species. It also indicates that each Tree Species is identified with a Leaf Pattern and has a general location in any number of Geographic Locations. For example, the saguaro cactus has leaves reduced to large spines and is generally found in southern Arizona and northern Sonora. Additionally, this figure indicates each Tree has an actual location at a particular Geographic Location. In this way, a particular tree could be classified as a saguaro and be located in Phoenix, Arizona.

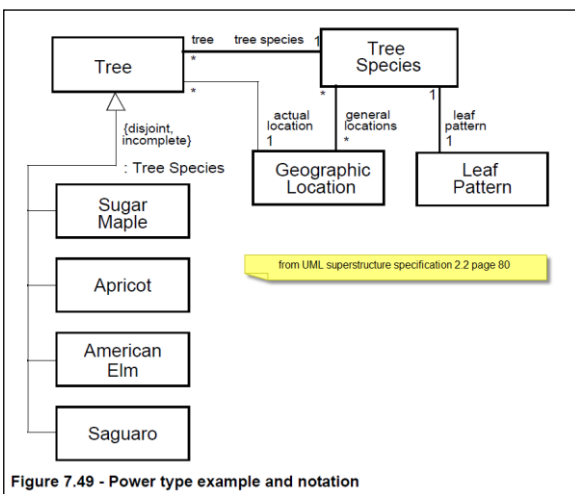


Figure 7.49 - Power type example and notation

Lastly, this diagram illustrates that Tree is subtyped as American Elm, Sugar Maple, Apricot, or Saguaro—or something else. Each subtype, then, can have its own specialized properties. For instance, each Sugar Maple could have a yearly maple sugar yield of some given quantity, each Saguaro could be inhabited by zero or more instances of a Gila Woodpecker, and so on. At first glance, it would seem that a modeler should only use either the Tree Species class or the subclasses of Tree—since the instances of Tree Species are the same as the subclasses of tree. In other words, it *seems* redundant to represent both on the same diagram. Furthermore, having both would seem to cause potential diagram maintenance issues. For instance, if botanists got together and decided that the American elm should no longer be a species of tree, the American Elm object would then be removed as an instance of Tree Species. To maintain the integrity of our model in such a situation, the American Elm subtype of Tree must also be removed. Additionally, if a new species were added as a subtype of Tree, that new species would have to be added as an instance of Tree Species. The same kind of situation exists if the

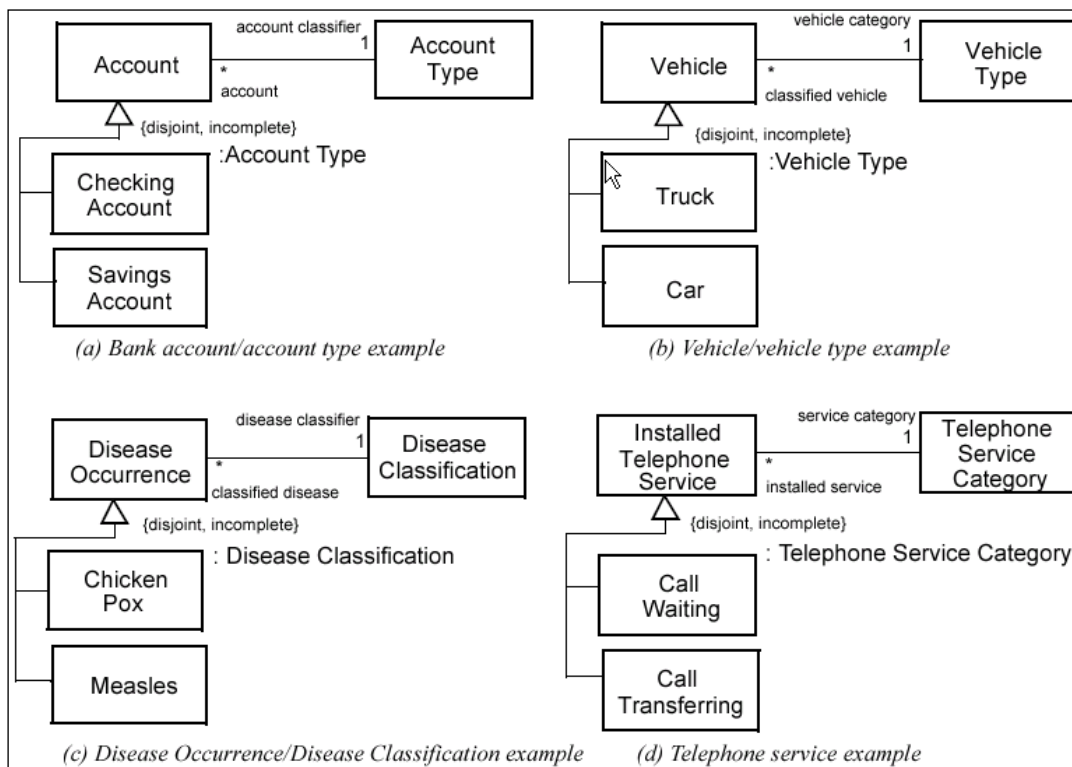
name of a tree species were changed—both the subtype of Tree and the instance of Tree Species would have to be modified accordingly.

As it turns out, this apparent redundancy is not a redundancy semantically (although it may be implemented that way). Different modeling approaches depicted above are not really all that different. In reality, the subtypes of Tree and the instances of Tree Species *are* the same objects. In other words, the subtypes of Tree are instances of Tree Species. Furthermore, the instances of Tree Species are the subtypes of Tree. The fact that an instance of Tree Species is called sugar maple and a subtype of Tree is called Sugar Maple is no coincidence. The sugar maple instance and Sugar Maple subtype are the same object. The instances of Tree Species are—as the name implies—types of trees. The subtypes of Tree are—by definition—types of trees. While Tree may be divided into various collections of subsets (based on size or age, for example), in this example it is divided on the basis of species. Therefore, the integrity issue mentioned above is not really an issue here. Deleting the American Elm subtype from the collection of Tree subtypes does not require also deleting the corresponding Tree Species instance, because the American Elm subtype and the corresponding Tree Species instance are the same object.

As established above, the instances of Classifiers can also be Classifiers. These same instances, however, can also be specific classifiers (i.e., subclasses) of another classifier.

When this occurs, we have what is called a *power type*. Formally, a power type is a classifier whose instances are also subclasses of another classifier. In the examples above, Tree Species is a power type on the Tree type. Therefore, the instances of Tree Species are subtypes of Tree. This concept applies to many situations within many lines of business. Figure 7.50 depicts other examples of power types. **The name on the generalization set beginning with a colon indicates the power type.** In other words, this name is the name of the type of which the subtypes are instances.

Diagram (a) in the figure below, then, can be interpreted as: each instance of Account is classified with exactly one instance of Account Type. It can also be interpreted as: the subtypes of Account are instances of Account Type. This means that each instance of Checking Account can have its own attributes (based on those defined for Checking Account and those inherited from Account), such as account number and balance. Additionally, it means that Checking Account *as an object in its own right* can have attributes, such as interest rate and maximum delay for withdrawal. (Such attributes are sometimes referred to as class variables, rather than instance variables.) The example (b) depicts a vehicle-modeling example. Here, each Vehicle can be subclassed as either a Truck or a Car or something else. Furthermore, Truck and Car are instances of Vehicle Type. In (c), Disease Occurrence classifies each occurrence of disease (e.g., my chicken pox and your measles). Disease Classification is the power type whose instances are classes such as Chicken Pox and Measles.



Labeling collections of subtypes with the power type becomes increasingly important when a type has more than one power type. The figure below is one such example. Without knowing which subtype collection contains Policy Coverage Types and which Insurance Lines, clarity is compromised. This figure depicts an even more complex situation. Here, a power type is expressed with multiple collections of subtypes. For instance, a Policy can be subtyped as either a Life, Health, Property/Casualty, or some other Insurance Line. Furthermore, a Property/Casualty policy can be further subtyped as Automobile, Equipment, Inland Marine, or some other Property/Casualty line of insurance. In other words, the subtypes in the collection labeled Insurance Line are all instances of the Insurance Line power type.

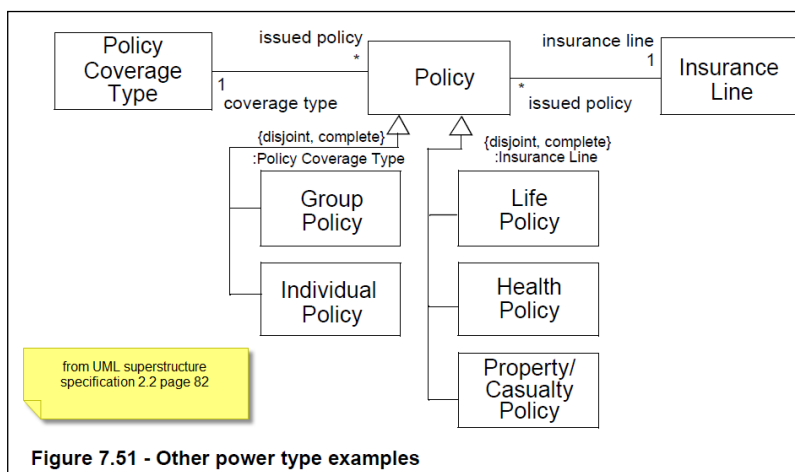


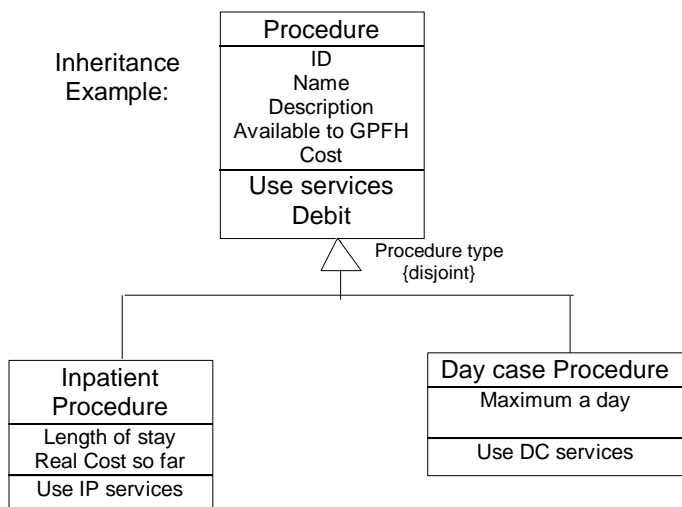
Figure 7.51 - Other power type examples

Power types are a conceptual, or analysis, notion. They express a real-world situation; however, implementing them may not be easy and efficient. To implement power types with a relational database would mean that the instances of a relation could also be relations in their own right. In object-oriented implementations, the instances of a class could also be classes. However, if the software implementation cannot directly support classes

being objects and vice versa, redundant structures must be defined. In other words, unless you're programming in Smalltalk or CLOS, the designer must be aware of the integrity problem of keeping the list of power type instances in sync with the existing subclasses. Without the power type designation, implementors would not be aware that they need to consider keeping the subclasses in sync with the instances of the power type; with the power type indication, the implementor knows that a) a data integrity situation exists, and b) how to manage the integrity situation. For example, if the Life Policy instance of Insurance Line were deleted, the subclass called Life Policy can no longer exist. Or, if a new subclass of Policy were added, a new instance must also be added to the appropriate power type.

----- end of extract -----

6.3.5 Generalization - Hospital Examples



The diagram opposite considers the idea of a procedure in a hospital. Procedures can, for this example, be of two types: inpatient and outpatient. However, both have a subset of identical characteristics and only differ on a few. We therefore create a 'procedure' class that is neither an inpatient nor outpatient procedure but contains the common characteristics of both. This procedure class is therefore something that does not exist in terms of instances (instances) but provides the basis to create other classes which would have instances such as day case hernia or inpatient varicose vein operation.

It is important to realise that any instance of Daycase Procedure will possess (i.e. **inherit**) the ID, name, Description, Available to GPFH, and cost attributes in addition to (i.e. inherited from) those attributes in the daycase procedure class. So it would end up with ID,

name, Description, Available to GPFH, cost and maximum a day attributes. The same situation applies to the operations. If an attribute or operation with the same name occurs in both the super and subtype then it is overwritten to follow that described by the subtype. You could also extend the above to have two generalisation sets say one concerned with inpatient/outpatient and the other NHS/private as shown opposite.

Let's recap:

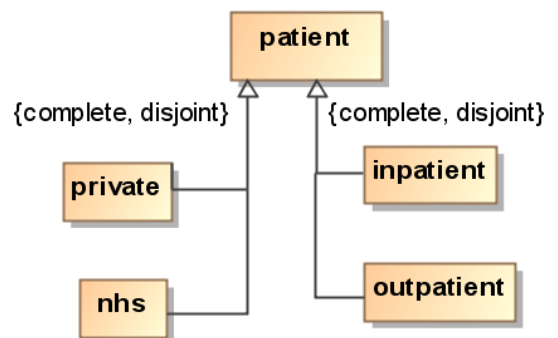
The hollow triangle in the above diagram indicates that both inpatient and day case procedures inherit characteristics from the procedure class. It can be read as '**is of type**'. The above diagram can therefore be interpreted as:

Procedure is of type inpatient OR day case

Note the '**OR**' in the above sentence. If we wished to replace it with an 'and' we would need to indicate it on the diagram. UML uses '{overlapping}'. The 'OR' situation is known as a **disjoint** situation whereas the 'and' situation is known as **overlapping** [previously OMT used a different symbol for this: a filled in triangle; UML does not].

The text beside the triangle 'procedure type' is the generalisation set name (called the **discriminator** in previous versions of uml) this can be an attribute whose value differentiates between the subclasses. Often this attribute is contained in the superclass.

The second example opposite shows a patient that may be one of four types, private inpatient or nhs inpatient or private outpatient or nhs outpatient. Notice that I have again interpreted the two separate specialisation subsets as being a AND situation.



Exercise 7. Generalisation/Inheritance

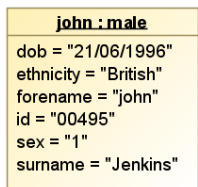
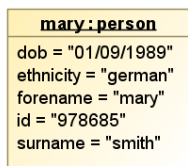
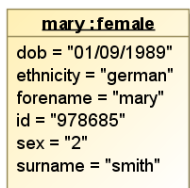
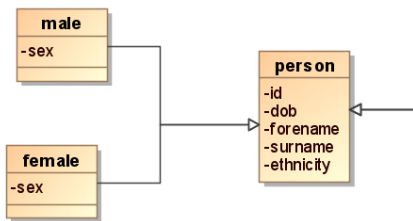
Draw class diagrams showing inheritance in one or more of the following situations:

- ◆ basic_cake_mix as superclass; define some subclasses
- ◆ Student as a superclass; define some subclasses
- ◆ Hospital_department unit as a superclass; define some subclasses
- ◆ A superclass of your own choice; define some subclasses

6.3.6 Misuse of inheritance - Enumerated data types

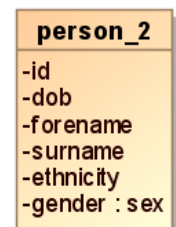
There is a danger that you will create a number of subtypes when all you need is what is called an enumerated data type, which was briefly mentioned in the last chapter.

Let's consider an example, say we have a superclass called PERSON and two sub classes MALE and FEMALE, the UML diagram below also shows three instances, one for each of the classes.

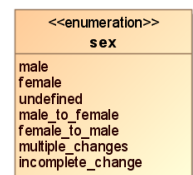


We note that both the subtypes have the same structure that is the same set of attributes, and also lets assume operations. The only difference is that for the female the SEX attribute = 2 and for the male =1. Instead it is more appropriate to just add a new attribute to the superclass which allows this value to be recorded, and I have called this new class PERSON_2 opposite.

Furthermore you can specify the valid set of values for the new attribute by either using an enumerated class



(indicated in UML by the <<enumeration>> title) or documenting it I prefer the documentation method.



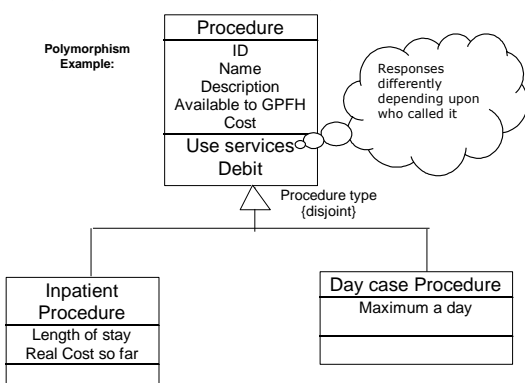
Exercise 8. Misuse of Inheritance

Please watch my Youtube video "inheritance (2) misuses and enumerated values in magicdraw" at:

<http://youtu.be/Gs5utYwH-Q8?hd=1>

6.3.7 Polymorphism

Polymorphism means that operations with the same name can behave differently in different classes. For example, consider the diagram opposite of various procedure types. The inpatient procedure class has an operation called 'use inpatient services' and the outpatient procedure class has an activity called 'use outpatient services'. It would be possible to have both the activities called 'use services' and the individual instances taking care of what actually happened. The advantage of this would be that the hospital manager instance would then only need to send one message to both instances 'use services' rather than a different one to each.

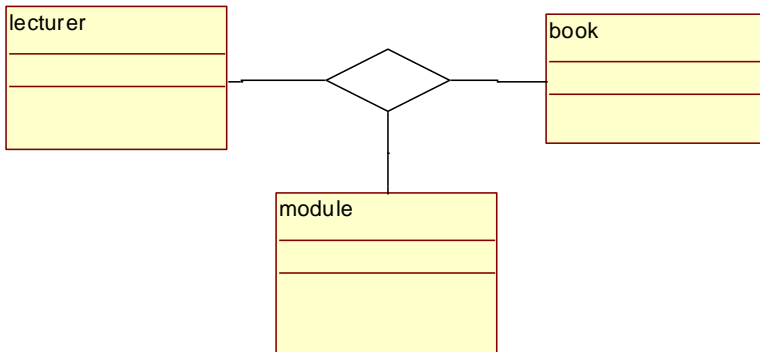


This is not so important to consider during the modelling stage but more important during the software development stage.

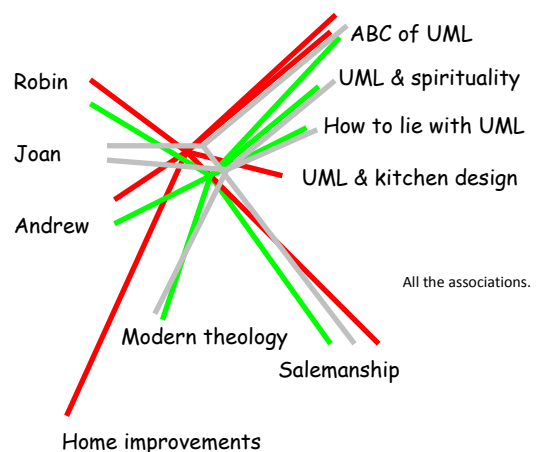
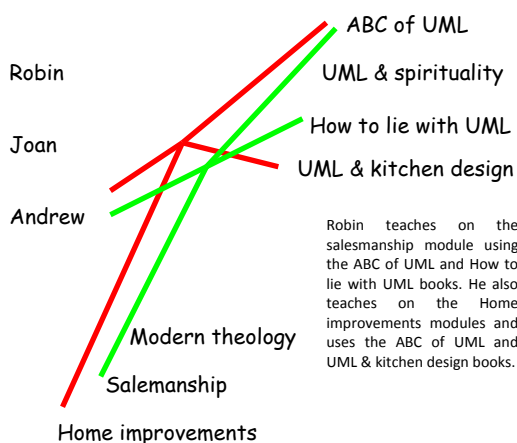
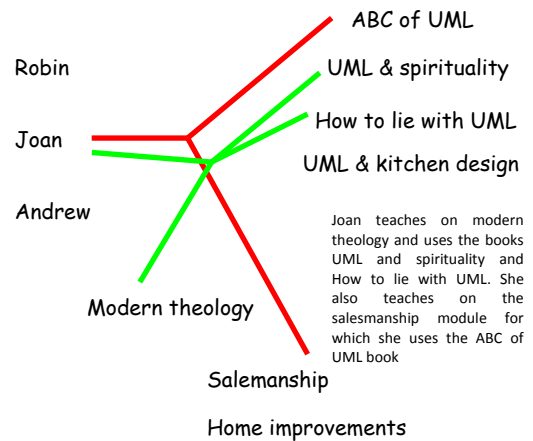
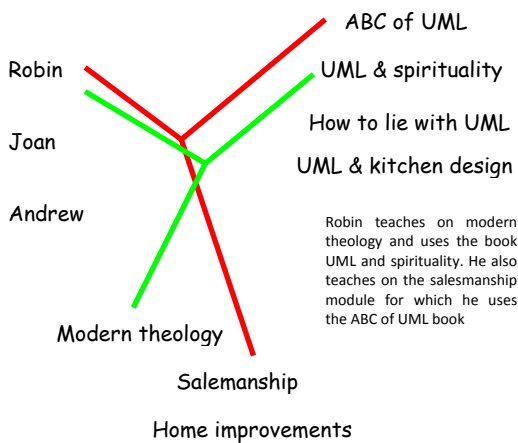
6.4 Ternary associations

Other than generalisation we have concentrated on associations which involve only two classes, however you can have associations which have more than two classes involved called **n-ary** associations, specifically one involving three classes is called a **Ternary** association.

Let's consider an example, below is shown a ternary association between Lecturer, Book and Module. What does this mean? Well it indicates that there is a clear association between a specific lecturer, book and module. For example with this diagram we could be able to discover that Lecturer Joan uses the book 'UML and spirituality' on the module 'modern theology'.



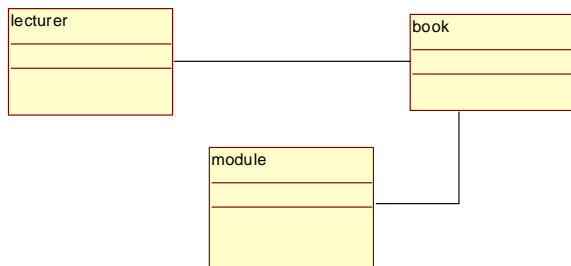
In other words we are defining a set of three specific values (L,B,M) in contrast to two specific values for an association involving two classes.



The first three **non** uml diagrams above indicate instances between Robin, Joan and Andrew who are three instances of Lecturer and other instances in the ternary association, the last one attempts to put them all together. Given this complexity you may feel inclined to simplify the situation and model it with two or even three binary associations. Is this possible?

6.4.1 Breaking Ternary associations into Binary ones

Most books recommend that it is sensible to convert ternary associations into a series of binary ones after one has considered the semantics of the particular situation you are modelling. For example we might assume after discussions with a group of lecturers that they ALWAYS recommend one or more books for a module. If this is the case then you could re-model the situation as shown on the left. But this means that any new lecturer instance that came along would need to play by the same rules. Is this reasonable?



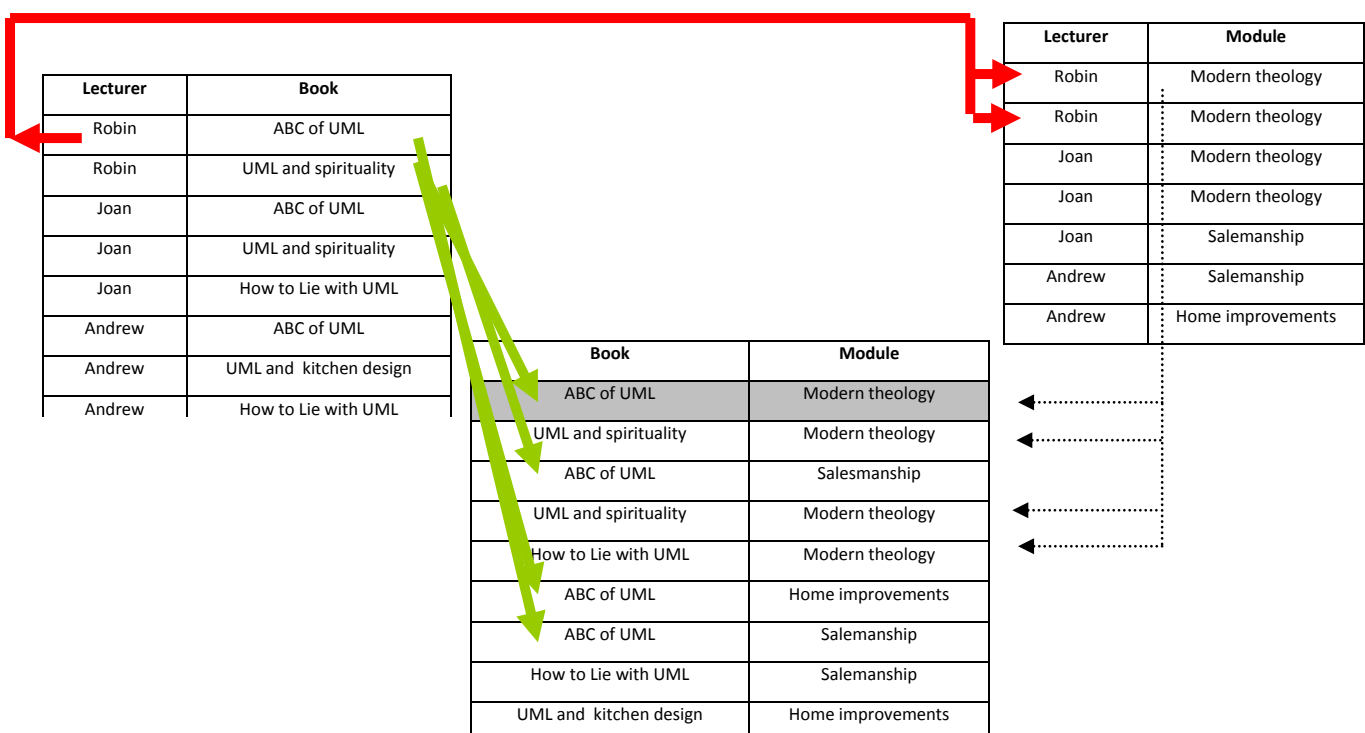
Also more worryingly by following through the new associations from Book to module, we do not really know which module which lecturer teaches on. Take for example, lecturer instance robin who is associated with

book instance ABC of UML, however the book instance ABC of UML is associated with module instances Modern theology, Salesmanship and Home improvements, we might mistakenly now assume Robin teaches these, very different from the ternary relationship!

| Lecturer | Book |
|----------|------------------------|
| Robin | ABC of UML |
| Robin | UML and spirituality |
| Joan | ABC of UML |
| Joan | UML and spirituality |
| Joan | How to Lie with UML |
| Andrew | ABC of UML |
| Andrew | UML and kitchen design |
| Andrew | How to Lie with UML |

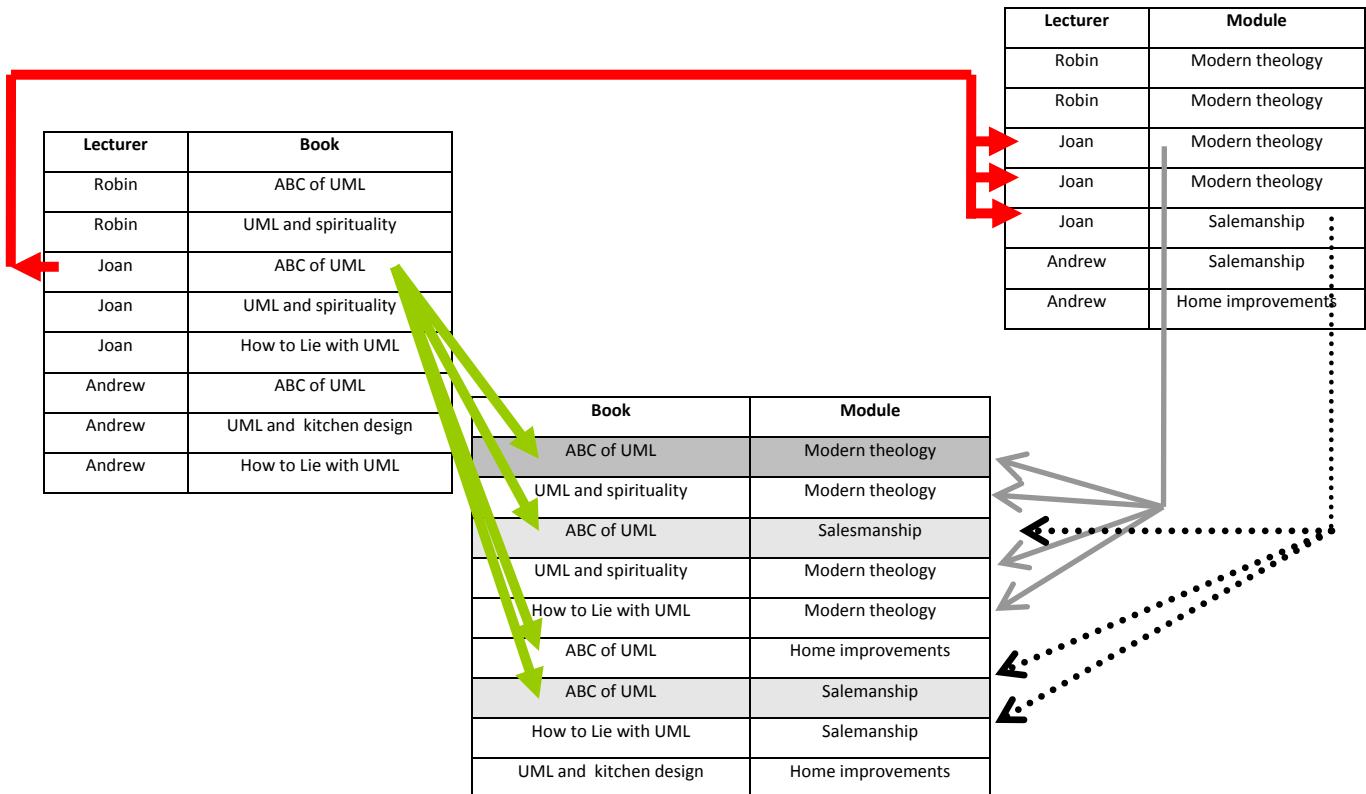
| Book | Module |
|------------------------|-------------------|
| ABC of UML | Modern theology |
| UML and spirituality | Modern theology |
| ABC of UML | Salesmanship |
| UML and spirituality | Modern theology |
| How to Lie with UML | Modern theology |
| ABC of UML | Home improvements |
| ABC of UML | Salemanship |
| How to Lie with UML | Salemanship |
| UML and kitchen design | Home improvements |

Can we mimic the Ternary association by adding another binary relationship between Module and Lecturer?



In the above example by considering the association between Lecturer and book and Book to Module (the green lines) we have four possible instances. In addition considering the association between Lecturer and module we have another four possible instances, However by considering the situation where both associations overlap (the shaded row in the Book, module instances) there is a unique association. So in this example we can identify the specific Book instance Lecturer Robin uses in the Modern theology module. But is this always the case?

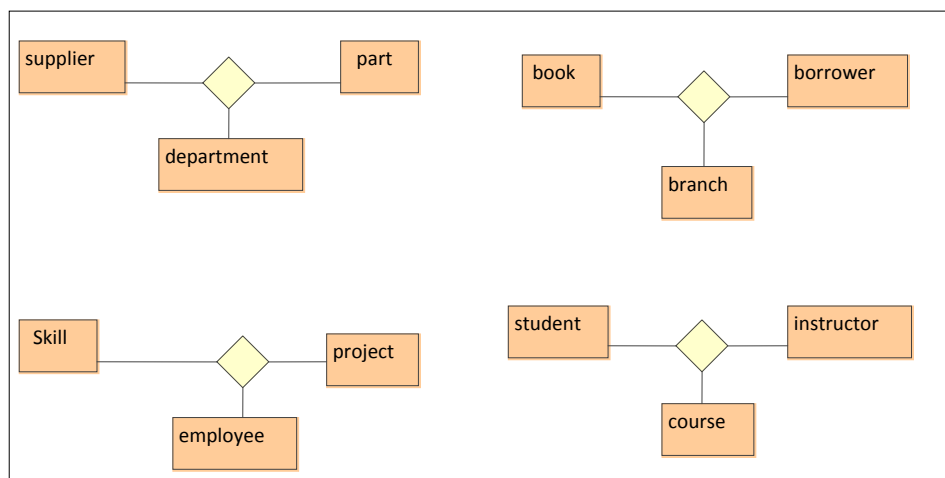
Taking another example, that of a instance of Lecturer, Joan lets see if we can identify a unique instance of book she uses for a particular instance of module, but simply can we identify which book(s) she uses for a particular module. From the diagram below it looks that this time we can't because there are two possible instances of the book ABC of UML associated with the Salesmanship module. In fact looking back at the ternary associations diagram we can tell from that, that the other instance is associated with lecturer instance Andrew. But it is impossible to derive this information from the binary associations we have set up below.



In conclusion it is possible to say that:

While ternary associations can be occasionally broken down into a series of binary associations the resulting semantics are subtly different to those of the original ternary association.

It is the responsibility of the modeller to ensure that the semantics of the final model truly reflect what the client is attempting to describe. On the right are four examples of possible ternary associations, but remember that in particular circumstances they might be broken down to binary associations.



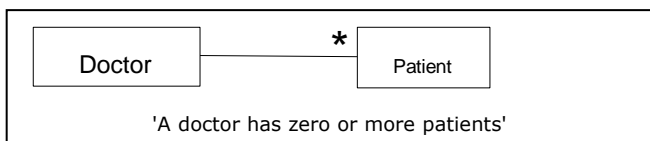
7 Mapping Associations to Databases

The instance modelling concept of association is pretty much the same as the database concept of a relationship. I will not bore you with the technical details of relational databases, which are based upon mathematical set theory. If you are interested you can consult the standard book on the subject - Date C J. 1995 (6th ed.) *An Introduction to Database Systems*. The relational database model was first described by Codd in 1970 who subsequently defined a set of rules based upon mathematical principles. His papers are referenced in Date 1995.

However what I will provide in the following subsections examples of the relationship between associations in UML class models and relationships in relational DBMSs.

7.1 One to Many Relationships

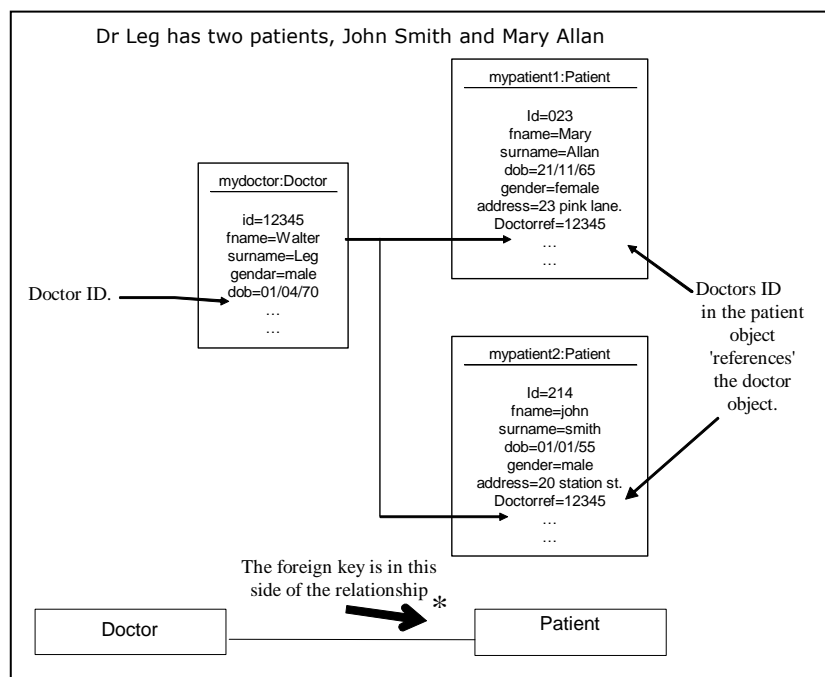
Consider the relationship between a doctor and patient. One doctor may have many patients. (Forget the situation of a patient having more than one doctor for the moment.) This can easily be drawn using UML:



But how do we turn this into something that can be represented as one or more tables (i.e. something that can be used by a computer)? The answer is that we make use of a concept called foreign keys.

If you do not understand the concept of relationships and foreign keys see: <http://www.robin-beaumont.co.uk/virtualclassroom/chap7/s3/dbcon2.pdf>

A foreign key, words like a pointer, it is a primary key field in one table that is embedded in another (or the same) table. In the example given opposite, doctor ID is the foreign key value in the patient instance. It allows the implementation in a DBMS of the association shown in the class diagram above. The diagram below shows exactly how.



Exercise 9. Converting Classes/instances to tables/records

Re-draw the instance instances of the one to many situation shown above as records in two tables.

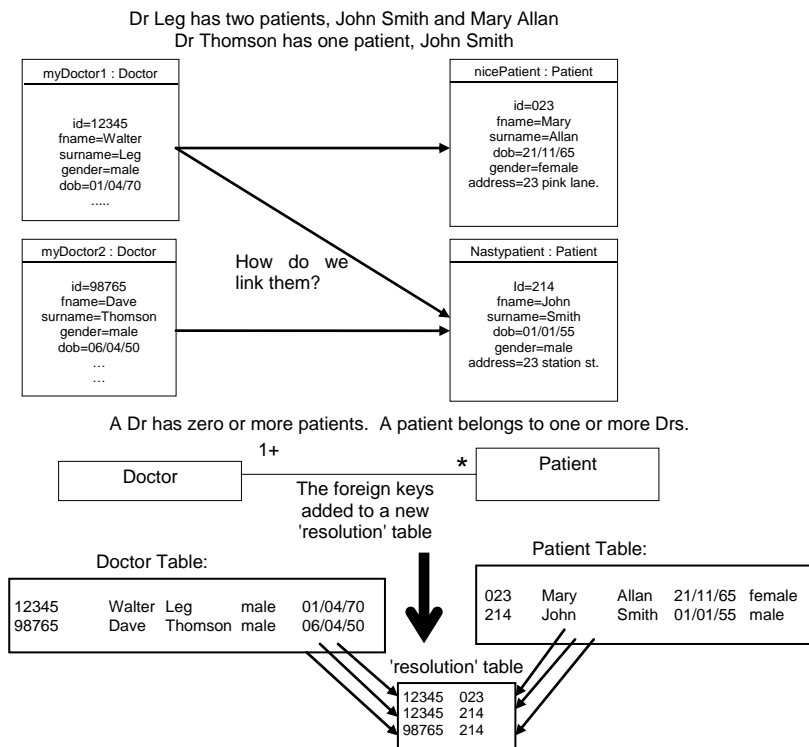
Important: in UML class diagrams you do NOT include foreign key attributes

7.2 Many to Many Relationships

Now for the more complex situation.

The one to many situation described above is rather restrictive as some patients may have more than one doctor. In other words the above model does not cater for all situations that could occur. The solution is to create a separate

table to link the two called a **resolution table**. This does not affect the instance diagram as it is not really necessary to show the table required in the database to solve the problem. However, some people do prefer you to do this, and it may help you create the database.



Exercise 10. MCQs

1. When mapping a class to a database, what does it become?

- A record
- A database schema
- An ERD
- A table definition
- An individual data item

2. When mapping a class instance, what does it become?

- A record
- A database schema
- An ERD
- A table definition
- An individual data item

8 The Meaning of associations (Semantics) and Business rules

People get very excited by attributing a great deal of knowledge and meaning to databases. In the 1980s a great deal of research took place developing '**semantic models**'. These were basically modelling techniques which had a wider range of modelling symbols, such as inheritance, which we have come across in instance oriented modelling. The standard semantics of models are entities (classes), properties (attributes or fields), relationships and subtypes (Date p.350). Therefore, the meaning -- or rather the semantics -- of the model are these elements, nothing more and nothing less. Date 1995 (p346 - 371), provides for this area of work an unusually lucid account of semantic modelling (i.e. what we have been doing in this chapter with the models).

The term **Ontologies** has been used by the Artificial Intelligence (AI) community to describe these more complex models, which also include a controlled vocabulary. You can find a good introduction to the subject in Mustafa Jarrar's PhD thesis at his web site however he has recently moved to another university (<http://www.jarrar.info/>) and unfortunately the content has gone his article *Data modelling versus Ontology engineering* provides an alternate source at <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.8114&rep=rep1&type=pdf>

Because no diagrammatic modelling technique can fully explain the complexity required in some databases, narratives are added which frequently take the form of sets of rules or constraints (e.g. if a patient is female and aged between 15 and 40 years, alert a doctor to take a cervical screening). They usually require programmers to implement, although a lot can be done with filtering out certain records and querying the records in some way. Such databases that contain these rules are frequently referred to as **knowledge bases**. Nowadays, though, the word is often used for any database regardless of its complexity.

As you have seen it is possible to derive UML class models from a set of narrative statements. Such statements have in the past been called **predicates** (Date p.97) but are now more commonly called **business rules**. We have been doing this in a very simple way by providing a set of sentences (a narrative) describing the class/instance model in each of the diagrams so far. So the UML multiplicity symbols along with the constraint expressions we have been specifying were just one way of specifying certain Business rules.

The 'meaning' of the data is also closely related to the very important idea of '**functional dependency**' -- basically, the dependency between various pieces of data (e.g. you can't have a blood result without first having a patient). This is closely related to a process known as normalisation, which provides a method to ensure the data is correctly structured in terms of functional dependency. These issues are considered in more detail in the Database section at <http://www.robin-beaumont.co.uk/virtualclassroom/contents.html>

Exercise 11. Business Rules

This is a optional exercise, I have included it because in the past several people have not seen how UML associations/constraints and business rules are similar. the links below are from Dr Terry Halpin's site (<http://www.orm.net/>) who has spent many years developing Object Role Modeling (ORM) which makes much use of what business rules and graphically looks a cross between uml class diagrams and ERD's.

Verbalizing Business Rules (part 1): PDF file (304K);
 Verbalizing Business Rules (part 2): PDF file (304K);
 Verbalizing Business Rules (part 3): PDF file (301K);
 Verbalizing Business Rules (part 4): PDF file (316K);
 Verbalizing Business Rules (part 5): PDF file (286K);
 Verbalizing Business Rules (part 6): PDF file (269K);
 Verbalizing Business Rules (part 7): PDF file (279K);
 Verbalizing Business Rules (part 8): PDF file (330K);
 Verbalizing Business Rules (part 9): PDF file (283K);
 Verbalizing Business Rules (part 10): PDF file (89K);
 Verbalizing Business Rules (part 11): PDF file (89K);
 Verbalizing Business Rules (part 12): PDF file (136K);
 Verbalizing Business Rules (part 13): PDF file (145K);
 Verbalizing Business Rules (part 14): PDF file (172K).
 Verbalizing Business Rules (part 15): PDF file (172K).
 Verbalizing Business Rules (part 16): PDF file (172K).

Temporal Modeling (part 1): PDF file (311K).
 Temporal Modeling (part 2): PDF file (317K).
 Temporal Modeling (part 3): PDF file (130K).
 Temporal Modeling (part 4): PDF file (613K).
 Temporal Modeling (part 5): PDF file (594K).
 Temporal Modeling (part 6): PDF file (630K).
 Temporal Modeling (part 7): PDF file (152K).
 Ontological Modeling (part 1): PDF file (107K).
 Ontological Modeling (part 2): PDF file (83K).
 Ontological Modeling (part 3): PDF file (138K).
 Ontological Modeling (part 4): PDF file (72K).
 Ontological Modeling (part 5): PDF file (109K).
 Ontological Modeling (part 6): PDF file (160K).

9 Encapsulation

Encapsulation has two different but related meanings. Firstly, it is used to mean the idea of an instance/class, in the manner we have been describing it above. That is, something that has both data and actions (Blair, 1989 quoted in Appleby 1991; Programming language concepts and paradigms). It can also be used simply to mean 'information hiding', meaning that other instances have to ask permission of the instance before being allowed to see its data.

However, both these uses have the same fundamental idea behind them: the data in an instance should not be directly accessible outside of the instance. For example, if another instance wishes to discover the waist measurement of the instance girl the requesting instance should not be able to just examine the value of the data item waist but request it via an activity of the girl instance such as 'request waist measurement'. Doing this, the girl instance has the ability to refuse. Often activities that provide some sort of information are called '**services**'.

Although this aspect of the instance paradigm is very important when it comes to actually writing the software, it is of little importance when beginning to model a system.

9.1 Packages

A Package in UML is just a container for a set of things that are considered to be related in some way. They are used to break up larger structures into more manageable bits. For example, if we tried to model the whole healthcare system we could divide it up into numerous packages possibly including medical services, financial management, research, community services etc.

Some software case tools use the term 'subject areas' which are basically the same as UML packages. Packages can be represented three different ways, as illustrated in the abstract from the UML 2.0. infrastructure document below for a package called 'Types'

Examples

There are three representations of the same package Types in Figure 95. The one on the left just shows the package without revealing any of its members. The middle one shows some of the members within the borders of the package, and the one to the right shows some of the members using the alternative membership notation.

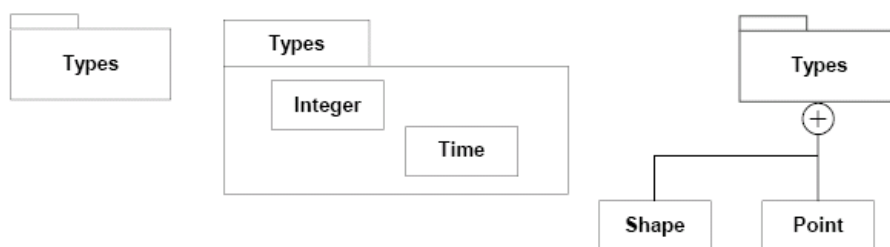


Figure 95 - Examples of a package with members

When using case tools, the user interface often demands that you place each diagram / class etc. in a particular package, obviously with a small project you can simply call the package 'the project'. However even with small projects where there are more than 20 classes it is often sensible to divide up the classes into different packages.

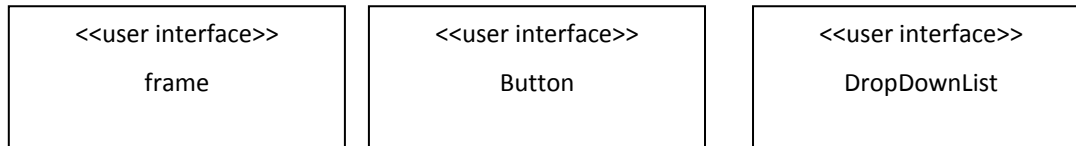
People often get unduly nervous about packages. Just think of them as folders for keeping similar UML elements; after all, the visual icon does look like a Microsoft folder icon.

9.2 Stereotypes

I have really only included this subsection to prevent you from using them!

Again this is a type of UML element which allows you to group others together. A stereotype is just a special type of class, again a method of grouping them together, we have already come across one example the **enumeration** stereotype. UML indicates a stereo type class by using the Stereotype indicator, that is the "<<stereotype name>>" at the top of the class.

The example below is of three 'user interface' stereotypes (from Pender, p.9):



UML has a number of pre-defined Stereotypes you can use. However as I said above I would warn you against using Stereotypes unless you are going to become a UML guru.

Exercise 12. Developing Class diagrams as a group exercise

It is all very well developing Class diagrams as a solitary activity. The real learning starts when you are doing it as a group.

Work as a group either face to face or using e-mail or a discussion board to carry out the following exercise.

Warning: Only choose **one** of the tasks below, and only spend a maximum of two hours on it.

- ◆ Develop an instance model for a work experience logbook.
- ◆ Consider developing a model for a prescription, which can consist of several different items.
- ◆ Develop a model for the area in which you work from two different perspectives.

At the end of the task, discuss amongst yourselves how you might check that you all have an agreed shared understanding of the model(s).

You might want to use a group working evaluation sheet.

10 Summary

In this chapter we have focused on the relationships between classes in terms of structure. We started by reviewing the symbols used by uml to define the common types of association and then went on to use them to extract the associations in a narrative by searching for verbs. We then went on to consider the more complex types of uml associations such as aggregation and composition (the modelling placebos).

Generalisation/inheritance was discussed in detail considering its correct and incorrect use demonstrating how an enumerated data type would be more appropriate in some situations. A few pages on ternary associations ended the section.

The relationship between uml associations and database tables, specifically the introduction of foreign keys, was described and the proviso that the concept is not usually shown in uml class diagrams emphasised.

Business rules and ontologies were mentioned in relation to uml associations and constraints, a connection which is often ignored.

The chapter ended by looking at a few of the more exoteric uml concepts such as packages and stereotypes, that people often become overly concerned, but are probably best not used in most models.

Now to check exactly how much you have learned, you should go back to the learning outcomes at the beginning of this document and tick off those with which you feel happy.

There are many areas of object oriented modelling that has not been covered in this document. One important aspect is the dynamic ("How") side of things. You will find details of this aspect in other documents at <http://www.robin-beaumont.co.uk/virtualclassroom/contents.htm>

I feel we have now covered most of the basics concerning UML class diagrams the secret is practice, practice, practice so to help you I have provided yet a few additional optional exercises below.

Exercise 13. Developing a Class diagram from a narrative

You should spend no more than 20 minutes on this task.

Draw the Class diagram which is implied from the information given below:

- ◆ A dental practice has one or more dentists and zero or more dental nurses.
- ◆ Each dentist has zero or more clients.
- ◆ Each client receives zero or more treatment sessions.
- ◆ Each session has a cost associated with it.

Exercise 14. Developing a Class diagram for a logbook

You should spend no more than 120 minutes on this exercise.

Imagine you have been asked to develop an electronic logbook for a doctor to allow you to analyse the following:

- ◆ Type of patients seen
- ◆ Type of procedures you were involved in
- ◆ Level of competence for each procedure
- ◆ Success of the interventions you were involved with
- ◆ Any other areas you might consider to be important

When you are sufficiently happy with your Class diagram, if at all possible enter it into a case tool. If not, use a drawing tool, or as a last resort use the drawing tools in MS Word.

If you are not a clinician, imagine that you are one. (You may want to talk to one after completing this task!)

Exercise 15. Developing Class diagrams for patient advice leaflets

The three following exercises are related:

1. Obtain several patient advice leaflets and see if you can come up with some standard section headings for each of the leaflets. Attempt to find advice leaflets from different sources and of different types.
2. Consider what type of advice leaflets you would have if you possessed all the patient advice leaflets which have been developed.
3. Attempt to amalgamate the headings you developed from the leaflets you obtained into a primitive type which would only need a few additional headings for any particular advice leaflet. (Look back at the section on inheritance.)

11 References

Alavi M Wetherbe J C 1991 Mixed prototyping and data modelling for information system design. IEEE software May 87 – 91. One of very few articles which uses an experimental design in place of the purely non experimental case study, to assess various methodologies.

Bennett S, Skelton J, Lunn K, 2005 UML: Schaum's outlines. ISBN 0-07-710741-1 £10.99 [A much improved second edition – no CD but what can you expect at this price]

Bertalanffy Ludwig von 1968 [but still published] General System Theory; Foundations, Development, Applications. George Braziller New York.

Blaha M Premerlani W 1998 Instance-oriented Modelling and design for Database applications; Includes UML. Prentice Hall

Blaha M, Rumbaugh J, 2005 Instance-Oriented Modelling and Design with UML (International Edition ISBN: 0131968599) Prentice Hall An e-book version is available (at: <http://www.safarix.com/> ISBN: 013132894-8 You can see quite a lot of the book from the free review sections.

Checkland P 1981 Systems thinking: systems practice. John Wiley An excellent introduction to the history of systems analysis as well as introducing Checkland's own 'soft systems' methodology.

Checkland P Scholes J 1990 Soft Systems Methodology in action John Wiley. This book includes a case study from the NHS.

Date C J. 1995 (6th ed.) An introduction to database systems

Fowler M Scott K 1998 UML distilled: Applying the standard instance language. Addison Wesley

Friedman A L Cornford DS 1989 Computer Systems Development: History, organisation and implementation. John Wiley. A wonderful book, excellent references, clear penetrating insights. etc.

Klein H K Hirschheim R A 1987 A Comparative Framework of Data Modelling Paradigms and Approaches. The Computer Journal 30 1 9 - 15.

Martin James 1989 - 1990 Information Engineering 3 volumes now published by Prentice Hall. The classic texts concerned with business process analysis. Very easy to read. £30 each volume. Moves from information strategy planning through business area analysis, technology impact analysis, to low level design. Firmly based in the business process paradigm.

Newman M. Rosenberg D. 1985 Systems analysts and the politics of organisational control OMEGA int. j. of mgmt sci., 13 (5) 393 - 406

Open University 1993 Relational Database Systems M866 (Five books; Introduction to database technology, The Relational Modal, Normalisation, Using SQL, SQL database management) [These are excellent resources with many exercises and clear concise explanations].

Pender A Thomas 2002 UML weekend crash course. Wiley publishing inc.

Reingruber Michael C. Gregory William W 1994 The Data Modelling Handbook John Wiley & Sons Chichester

Rumbaugh J, Blaha M, Premerlani W et al 1991 Instance-Oriented Modelling and Design. Prentice Hall. See Blaha, Rumbaugh J, 2005

Schmuller Joseph 2004 Sams Teach Yourself UML in 24 hrs ISBN 0-672-3260-40 Third edition. [This new edition has a CD containing a pdf version of the book and a Case Tool called Poseidon (no time limit). Costs around £20]

Tsang C H K, Lau C S W, Leung Y K, 2005 Instance-Oriented Technology ISBN: 0071240462 McGraw-Hill Education [rather overly concerned with programming for this course however it does come with a CD of the case tool VP-UML but the book is not on the disk]

Joubert M The Use of Conceptual Graphs for Knowledge Bases, Customisation and Actual Data Organisation, Unpublished project working paper (AIM NUCLEUS)

Wood-Harper AT Fitzgerald G 1982 A Taxonomy of Current Approaches to Systems Analysis The Computer Journal 25 1 12 - 16.

Yourden 1989 Modern structured analysis Prentice hall. One of the standard texts on structured analysis again from the business process paradigm.

12 Appendix - OMT

Types of Association Based on Fowler & Scott 1997 p59

