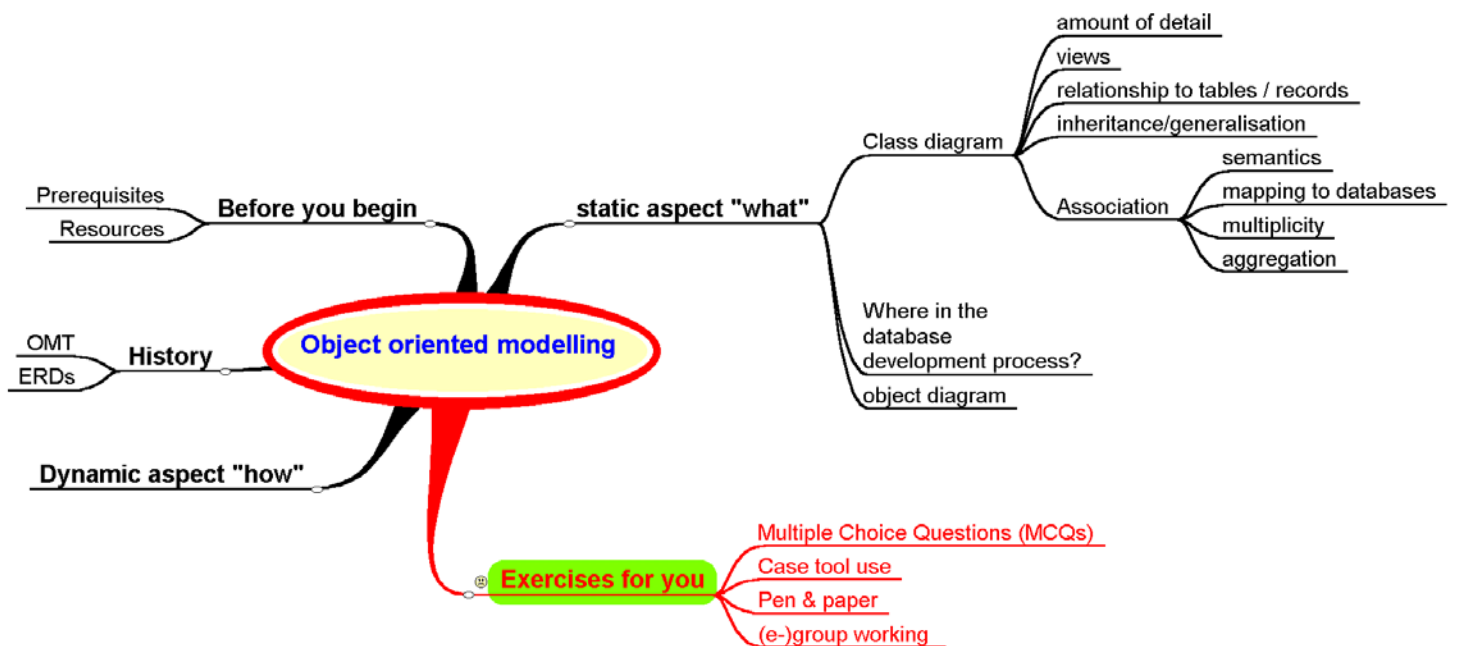


An Introduction to Class Modelling using UML (Unified Modelling Language)

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

Date last updated: Friday, 29 May 2009

Version: 5



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

Contents

1	Before you start	4
	Required Resources	4
2	Learning Outcomes	5
3	Introduction	6
4	Object Oriented Approaches and UML	6
5	Class instances	8
6	Classes	9
6.1	Developing Databases – An example of where Class Diagrams might fit into the Process.....	12
6.2	The Relationship between Classes and Objects to Databases.....	13
6.3	Class Diagrams and Amount of Detail Shown.....	14
6.3.1	Views.....	14
6.4	Activities/Operations/Methods	15
7	Associations	15
7.1	Multiplicity.....	17
7.2	Aggregation and Composition.....	18
7.3	Ternary associations	19
7.3.1	Breaking Ternary associations into Binary ones	20
7.4	Recursive associations	22
7.4.1	Association End Names	22
8	Mapping Associations to Databases	24
8.1	One to Many Relationship.....	24
8.2	Many to Many Relationship.....	25
9	The Meaning of Relationships (Semantics)	28
10	Inheritance	28
10.1	Generalisation Sets	29
10.2	Constraining Subclasses.....	29
10.3	Inheritance - Hospital Example	31
11	Encapsulation	32
12	Polymorphism	33
13	Larger models and using Case tools	33
13.1	Packages.....	33
13.2	Stereotypes	34
14	Exercises	35
15	Summary	37
16	References	37
17	Web Links	38

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

1 Before you start

This document assumes that you have the following knowledge and skills:

Skills/ knowledge:

- That you have worked through the document “An Introduction to ERDs” at <http://www.robin-beaumont.co.uk/virtualclassroom/contents.htm>
This will have given you the experience of creating ERDs, using a Case Tool and understanding some of the issues associated with modelling.

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.htm> and follow the links
- 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 System Architect and MagicDraw. You can download a community version of MagicDraw at <http://www.magicdraw.com/>. 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 enable you to do this you need to contact your course administrator who has the codes to unlock the personal edition of MagicDraw. The document “An Introduction to ERDs” provides details of other websites where you can download alternative software.

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

2 Learning Outcomes

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

Learning outcome	Tick box
Be able to describe the main characteristics of object oriented modelling	<input type="checkbox"/>
Be able to provide brief details of the relationship between OMT and UML	<input type="checkbox"/>
Be able to describe briefly the history of UML	<input type="checkbox"/>
Be able to describe what an object is, as used in object oriented (OO) modelling	<input type="checkbox"/>
Be able to describe the three parts of a class	<input type="checkbox"/>
Be able to describe the two parts of a class instance (often called a object)	<input type="checkbox"/>
Be able to describe the relationship between attributes and data items	<input type="checkbox"/>
Be able to explain the difference between a class instance and a class	<input type="checkbox"/>
Be able to identify and draw classes and instances using the UML notation	<input type="checkbox"/>
Be able to explain the relationship between classes, objects, tables and records	<input type="checkbox"/>
Be able to describe the varying amount of detail that can be displayed in a class diagram	<input type="checkbox"/>
Be able to discuss the concept of “view” as it relates to a class diagram	<input type="checkbox"/>
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/object models are implemented in a database	<input type="checkbox"/>
Be able to describe briefly the concept of semantic modelling	<input type="checkbox"/>
Be able to describe, and provide an example of, inheritance	<input type="checkbox"/>
Be able to describe the various ways of modelling inheritance	<input type="checkbox"/>
Be able to demonstrate the usefulness of 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 Introduction

The process of designing any system, whether it be computer, paper or person based or a mixture of all three, consists of specifying two important aspects, amongst other things: the **what** (data - structure) and the **how** (what it does - behaviour). Considering the **How** aspect, a system is worse than useless if it either is difficult to use (e.g. for entering or retrieving information) or hinders rather than facilitates working practices. For example, if a system is planned to be used in a medical consultation it should be easier rather than more difficult to prescribe and allow the users to collect data in the way they would normally. While this **How** aspect is as important as the data side of things, we will concentrate on the 'what' side of things in this document.

You will be familiar with ERDs, a graphical technique for specifying the **what**. In this document we will be looking at a more advanced diagramming technique to produce Class/ class instance (object) diagrams which consider the **what** aspect in more detail. These diagrams are one aspect of 'object oriented' modelling techniques that have developed over the last few decades. Basically the Object oriented approach allows more complex models to be developed than previously possible using such techniques as ERD diagramming. Before we consider Class and class instance (object) diagrams in detail, we will take a quick look at object oriented modelling techniques in general and UML.

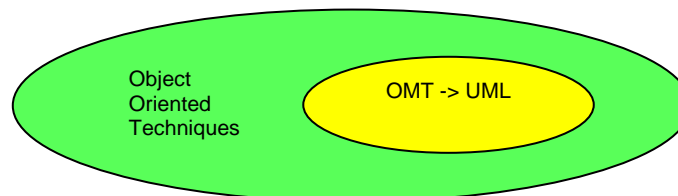
4 Object Oriented Approaches and UML

In the early 1990s Rumbaugh as well as M Blaha et al developed a modelling process called **OMT** (Object Modelling Technique). It has been documented in a book by them called *Object-Oriented Modelling and Design*, Prentice Hall 1991. This book has taken on the status of a classic now and is still available although they have just published, what can be considered to be, an update to this in 2005 – *Object Oriented Modelling with UML* (I believe an e-book version is also available for this version).

So what are Object Oriented Modelling techniques and how do they relate to this thing call UML?

Object oriented modelling relies heavily upon the following five ideas (concepts), which allow us to model aspects of the world in a logically consistent manner. Notice this is much wider than just database modelling:

- Classes and Objects
- Association
- Inheritance
- Encapsulation
- Polymorphism



In this document we will concentrate mostly on the first two aspects, although we will have a glance at the others in passing. So how does UML relate to Object Oriented Modelling? The answer is basically historical.

In 1998, Rumbaugh joined forces with Grady Booch and Ivar Jacobson, who also have their own Object Oriented Modelling languages, to create a **Unified Modelling Language (UML)**. That year saw a burst of activity with several books being published describing UML (Fowler & Scott 1998, Blaha & Premerlani 1998), UML not only subsuming Rumbaugh's OMT but also expanding it with new diagrams. Since this time UML has gone through a number of revisions with the most recent being version 2 (Around December 2004). I have therefore decided only to discuss the slight differences that exist between the two occasionally in this document. The latest version of UML can be found at <http://www.omg.org/>.

UML is definitely the "in thing" and if you look in any computing/ modelling / system development section of most large bookshops you will find a row of books about UML. Three that I would recommend are:

Joseph Schmuller 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]

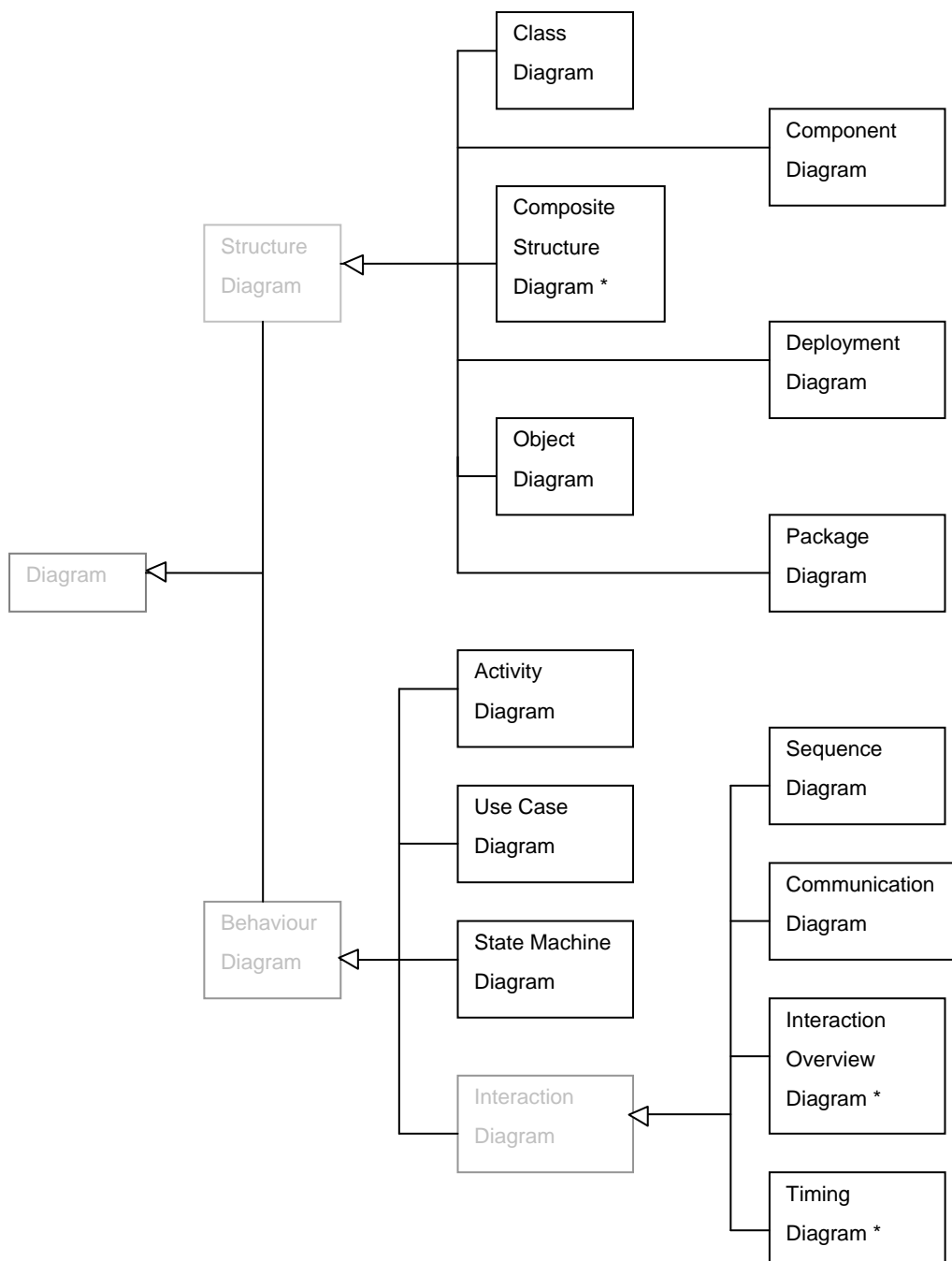
Simon Bennett, John Skelton, Ken Lunn, 2005 UML: Schaum's outlines. ISBN 0-07-710741-1 £10.99 [A much improved second edition – no cd but what can you expect at this price]

Thomas A Pender: 2002 UML Weekend crash course. ISBN 0-7645-4910-3 [Also contains a CD with a pdf version of the book and a 15 day demo version of System Architect version 8.5 Costs about £15 – 20]

You can get second hand copies of the above books for far less by looking on www.addall.com, www.Amazon.com and www.amazon.co.uk

UML provides a standard set of tools (most of us would call these diagrams – but the UML people get upset if you reduce them to such things!) for analysing and developing a model using an Object Oriented Approach. UML is not a method, you are free to use which tools (diagrams) you feel are appropriate and at the appropriate points you decide during the process. However this does not mean there is no relationship between the various diagrams; one of the most important aspects of a CASE TOOL is how it manages the relationships that exist between them as specified in the UML. Few if any modellers use all the diagrams; it depends upon the specific project and the personal preferences of the modeller.

Fowler 2004, p.12 has a nice diagram showing the various diagrams in UML (version 2) taken from the formal UML specification document; the greyed out boxes represent the classification of the diagrams. In UML version 2 there are 13 different diagrams, including 3 new ones (marked with an asterisk):



In the rest of this document we will focus on the Class diagram, probably the most important diagram in UML. But to help you understand what classes are we will look briefly at class instances (Objects) first.

If you want a quick overview of UML and the various diagrams see:

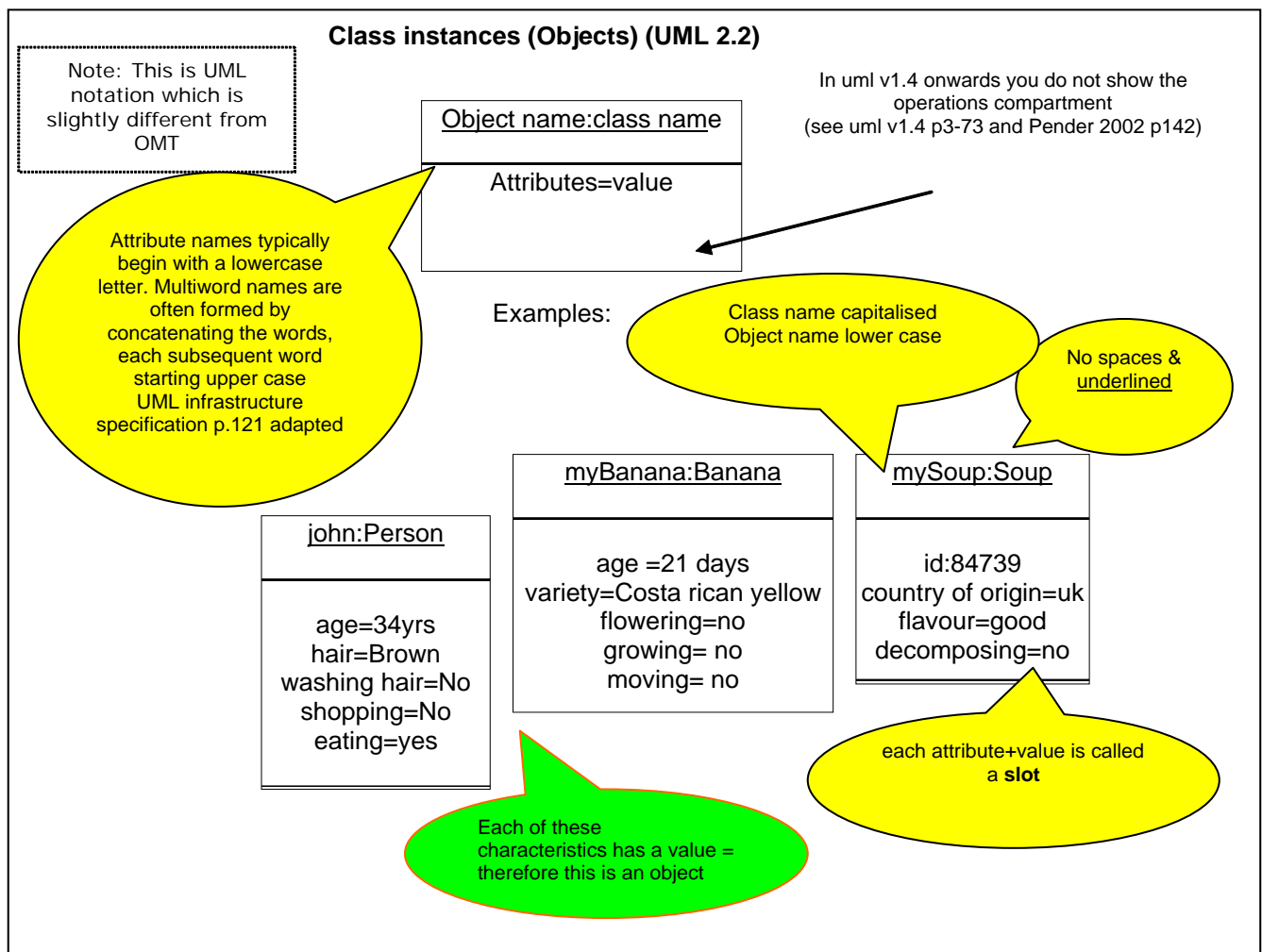
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/intro_rdn.pdf or
<http://www-128.ibm.com/developerworks/rational/library/769.html>

5 Class instances

So what is a class instance in the context of 'object oriented modelling'?

A class instance (often called a object in UML) is best thought of as a definable thing with a number of facets that results in us perceiving it as something that is different from something else. In biology people often talk about something having form and function. Taking this a little further, one can think of the 'descriptive' side of things as being **data** (e.g. a person having a name, age, hair colour and address etc) and the 'function' side of things as being **activities** (e.g. washing hair, giving birth, going shopping etc). Moreover, the data (structure) and activities (behaviour) are inextricably linked in our conceptualisation of the object. If the object should change either its activities or characteristics it exhibits, we feel distinctly uneasy. In other words, when we conceptualise an object in the world, we do not naturally divide it out into these two aspects. To do this requires skill.

For example, when we see something we like, we tend to think of actions as well as the class instances characteristics (data). Taking this further, when one thinks of a banana or a bowl of soup, not only do both have pleasant characteristics such as colour and smell but they also have pleasant actions associated with them such as the soup being made (created, the default operation all objects possess) and the banana growing.



In the context of 'object modelling' a class instance is therefore something in the investigator's mind that usually possesses a recognisable number of both characteristics (data consisting of data items) and actions. As in all areas of computer science, one word is never enough; UML use the word '**attribute**' instead of data or characteristic to mean the same thing, and the words '**operations**' or '**methods**' to mean activities/actions. I will use the words **attribute** and **operation** in this document.

The diagram above shows how class instances are drawn in UML. Each class instance is divided into two sections:

- The top box gives the name of the class instance along with its class name (more about that below).
- The bottom section lists each attribute along with its value.

All the above examples of class instances (objects) indicate that the attributes have values suggesting that the class instance exists in some way and because of this it often means class instances are called 'real world things' but this can be misleading.

From the above description you may be feeling that the concept of a class instance is very similar to that of an Entity Instance, which you considered in the document concerned with ERDs. In fact you would not be far from wrong; however, the class instance concept also has the concept of Operations within it, they are just not shown in the object diagram. To learn about these operations we need to move on to the next Object oriented modelling concept listed at the start of this document, that of the Class.

6 Classes

The best way to think of a class is to think of a blueprint of some type. It is often called a concept, although philosophers argue about the differences. Many theorists have provided complex definitions of a class or entity type, which is a similar idea, but none are complete or infallible.

Banana
age variety flowering growing moving
flower() grow() move()

A Class

Hopefully by the end of this section you will be able to conceptualise what a class is in this context even if you still won't be able to provide a definition for it.

Chen, one of the early writers (circa 1970), felt that the task of class identification (he was talking about 'entity types' at the time, as object modelling had not been developed) was best left to the individual who was within, and who knew most about, the system to be modelled. This suggests that classes are partially dependent upon the person who is identifying them, and this will become clear as you work through this section.

What then is a class? A class can be thought of as an abstraction from the object. Notice that in the examples concerning objects a single banana was

described. The object 'mybanana' is said to be of type class banana. Similarly John is said to be of class men and "my leek and ham soup" of class soup. Notice that while the class men could exist without any objects of that type, the reverse is not true. We say that an object is an '**instance**' of, or an '**instantiation**' of a class. This may appear rather pedantic but it is very useful in modelling as it provides a method of classifying groups of things often in very useful ways.

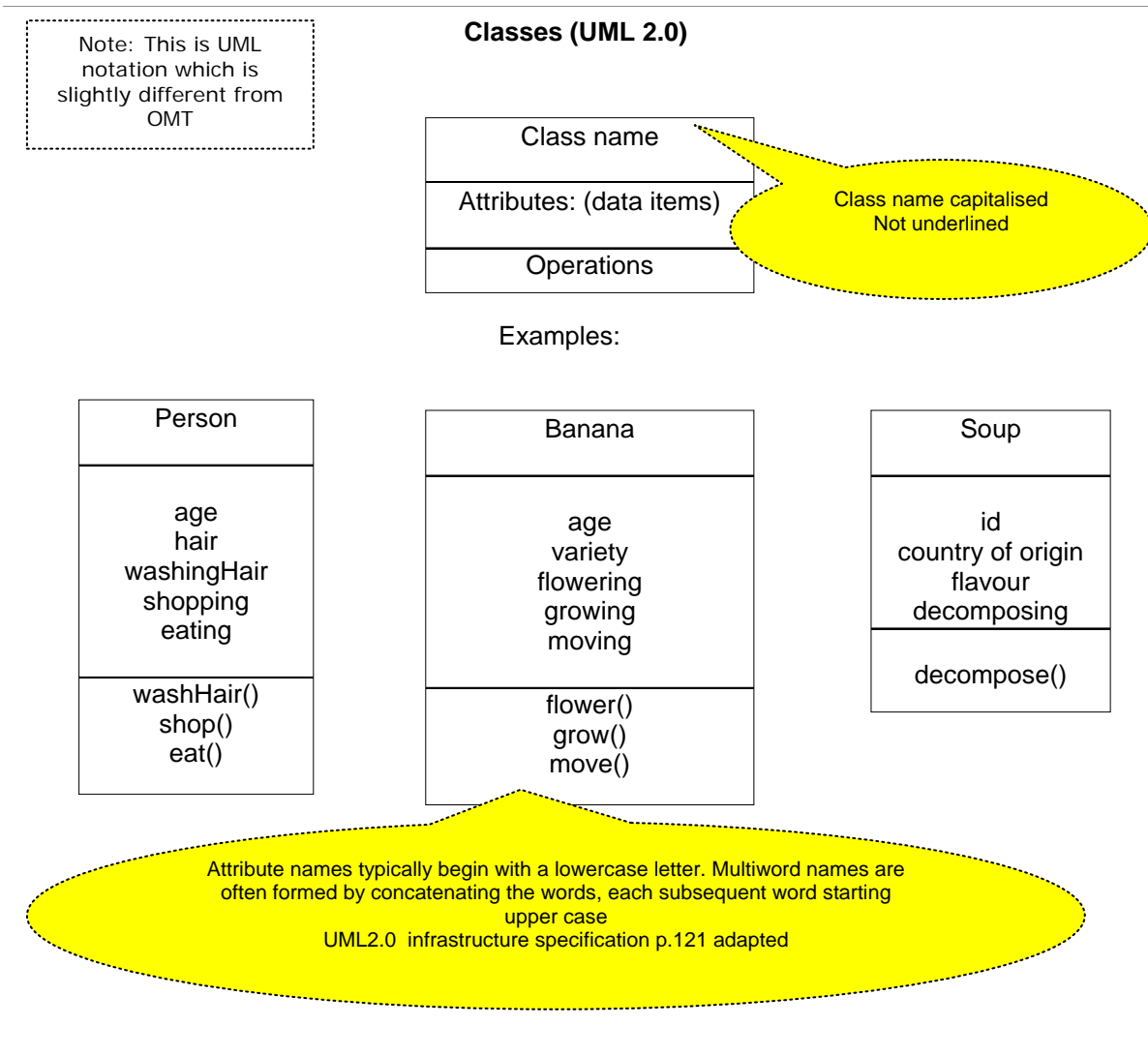
Basically:

- **Class = Template**
- **Object = Instance**

Computer scientists, whenever possible, use diagrams or symbols rather than words. This is because words are considered to be the most ambiguous form of communication, or more probably because most computer scientists are notoriously poor at English. The diagram opposite shows how UML (and OMT) would possibly represent several of the classes described above:

In the diagram each box represents a class. The box is divided into three sections. The top section of the box provides the name of the class, usually a noun. The next section down provides details of the data (attributes or data items) in the class while the last and final section provides names of the activities (methods) that make up the class. While the first two examples -- Person and Banana -- are relatively easy to define in this way, the third is more difficult. This is probably because Soup has a large number of attributes but has very few activities. The only one I could think of, besides the generic 'create' which all classes have, how else would you create instances? is the process of decomposing when it's left too long, possibly 'heat up' might be another. Not all classes have actions (other than the generic 'create' and 'destroy'), but all must have a name and something that uniquely distinguishes them from another class. Because objects are just instances of a particular class, these rules apply to them as well.

In this document we will concentrate on class diagrams rather than object diagrams; being more general, they tend to be more useful. However, object diagrams are useful if you want to investigate a particular situation.



You may be thinking that classes and Objects (instances) are the same as Entity Types and Entity Instances however Classes and objects are far more complex because they possess operations as well as attributes.

For example in the discussion concerning entity types and instances on page 10 in the ERD document (http://www.robin-beaumont.co.uk/virtualclassroom/chap11/s9/erds_1.pdf) we re-consider the diagrams to represent them as objects and classes now.

Doctor = Class

Dr Dow smith (GMC number 39200456) born 19/12/1956 etc. currently examining Miss smith, considering diagnosis of severe self-centredness = object (i.e. class instance)

Why is this a class?

Because it as a name

Because it has a set of unique attributes i.e. surname, forename, date of birth, GMC number, salary, gender etc.

And a unique set of operations i.e. examine, question, diagnose, request test etc.

Why is this a class rather than an instance?

Because it possesses these attributes and operations but they do not have specific values

Why is this an instance (=object) of Doctor Class?

Because it has the **same set of attributes** as the doctor class i.e. surname, forename, date of birth, GMC number, salary, age etc.

And each attribute has a specific value

Because it has the same set of operations as the doctor class

And each operation has a value.

Another way of thinking about the bond between a class and instance (=object) is to think of the class as being the column headings (= attribute + operation names) for a number of rows each of which is an instance of the class.

Doctor class with 3 instances

Doctor							
surname	forename	date of birth	GMC number	salary	gender	examine	Request test
smith	Dow	19/12/1956	39200456	89K	male	yes	Blood for culture
Coates	Jill	03/05/1966	5748337	67K	female	no	Knee x ray
Worsley	Alan	11/07/1970	578493	80K	male	no	Barium enema

Six attributes shown

Two operations shown

Exercise 1. MCQs

1. Which one of the following is an example of a class:

- Robin Beaumont
- Carrot
- Date's book *An Introduction to Database Systems* on my shelf at home
- Spike from the TV series "Buffy the Vampire Slayer"
- My backup disk (in the second drawer of my desk)

2. Which one of the following is an example of an instance of a class (object):

- Course Manager
- Health Informatics courses
- Women
- Tony Blair
- Religion

Exercise 2. Classes and Objects

Part A

Consider your job to be a class. If you do not have a paid job, you may be a student for example, then use that instead. List two of your (clinical) activities along with the information about them which you feel would be useful to collect. If it helps, consider it in the context of a learning log book of some sort.

Part B

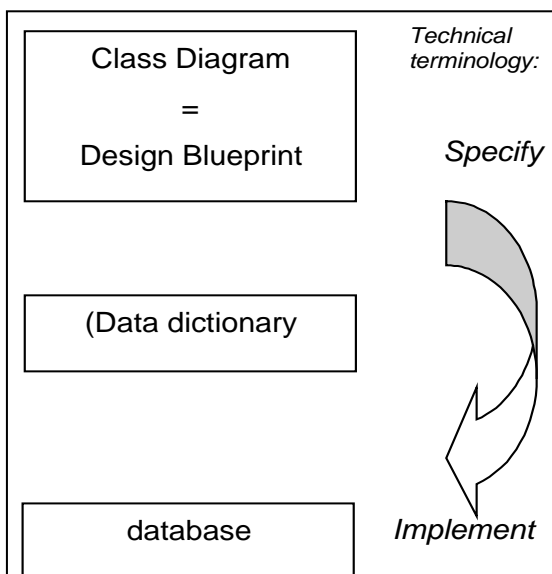
Draw a diagram that represents the following classes: community nurse, GP, clinical manager in a hospital, day care coordinator, director of finance in an acute trust (hospital outside the UK), patient, yourself and your boss.

Part C

Draw an object diagram that represents an instance of some of the classes you described in the above tasks.

6.1 Developing Databases – An example of where Class Diagrams might fit into the Process

UML is often used, BUT NOT ALWAYS to develop computer systems and frequently these are databases. When people first start developing databases they always make two fundamental errors. Firstly, they rush to a computer to play with the DBMS (e.g. Access). Secondly, they believe they can create a perfect database by specifying a model without creating a prototype of some sort.



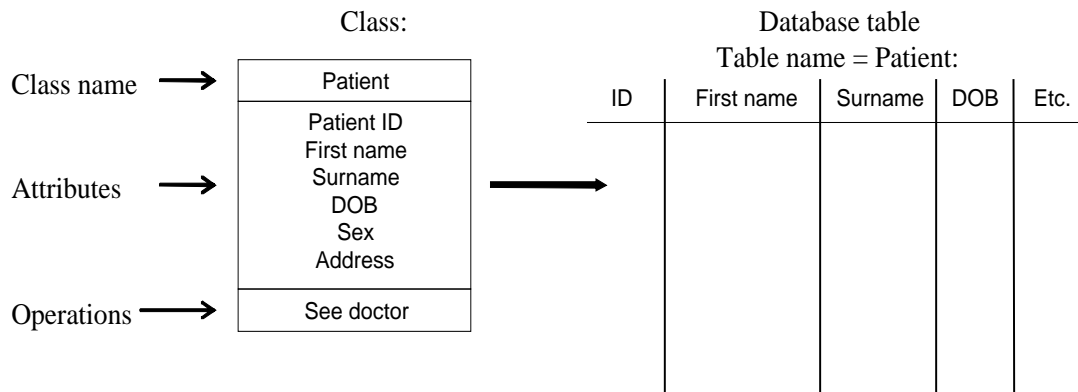
It is vitally important to get the design right, but this involves many revisions resulting partly from feedback from people using the actual database. For each iteration the Class diagram(s) allow you to create a blueprint (model, specify, design, etc) of the data, and the DBMS allows you to create (implement, develop, etc) the actual database.

In the diagram to the left, the data dictionary is shown as a half-way house between the 'high level' object model(s) and the actual database. Depending upon your inclination it is often not necessary to produce a data dictionary as a detailed description of each object is adequate; it is basically up to you and whomever you may be working with. (See Blaha & Premerlani 1998 for more detail.)

Again I must reiterate Class diagrams are not only used to develop databases!

6.2 The Relationship between Classes and Objects to Databases

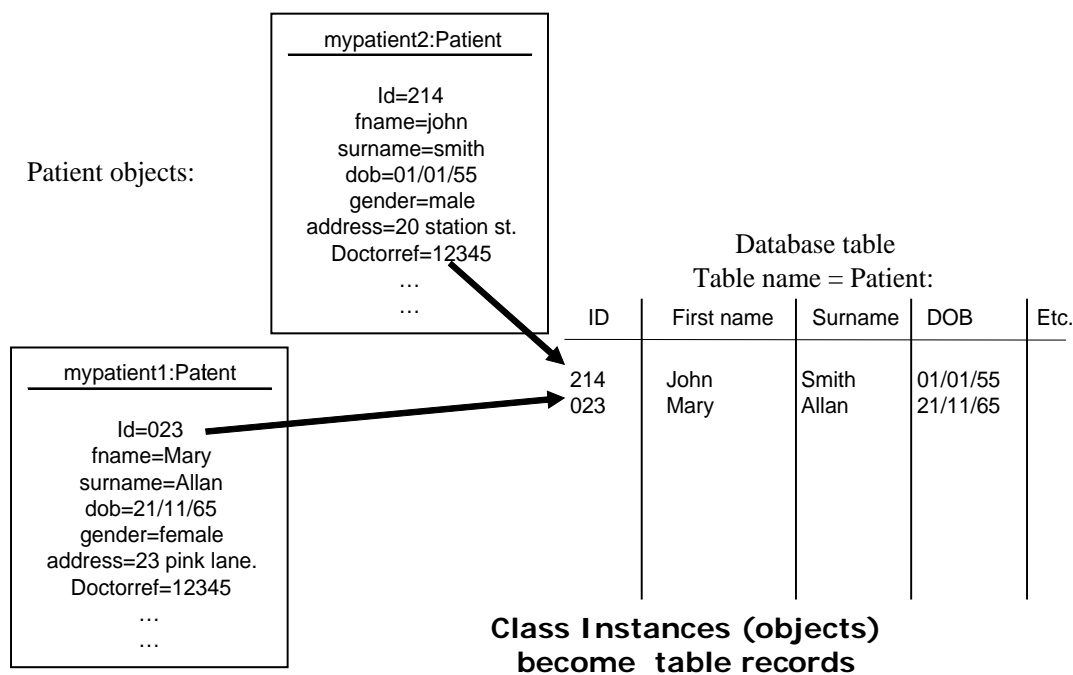
If we accept the object view of the world, databases themselves are also simply a collection of objects. More specifically a table is simply a way of allowing the computer, via a piece of software called a DBMS (Database Management System, e.g. Access), to store attribute and instance details of objects. This all sounds rather abstract, so I will now provide some examples.



Class name becomes table name (not always)
Attributes in class model = fields in database

The above diagram demonstrates the relationship between a class and a DBMS table structure. The attributes in the class diagram become fields in a table where the table name is the same as the class name.

Taking this one step further, an object (class instance) is equivalent to a record in a table in the database.

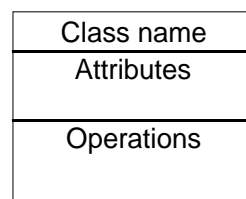


Class Instances (objects) become table records
Value in object model = data item in database

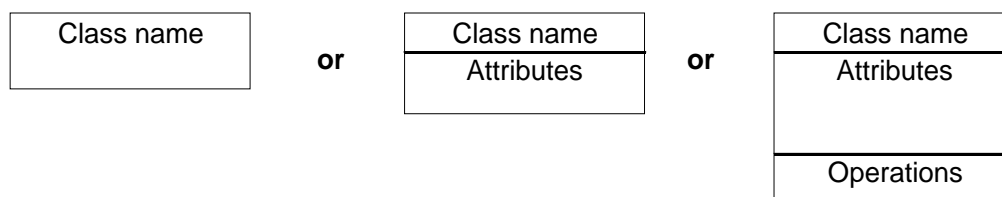
Exercise 3. Linking Classes and Objects to Databases

Spend a few minutes taking some of the classes / objects you have identified in the previous exercises and draw a similar diagram to the appropriate one above.

6.3 Class Diagrams and Amount of Detail Shown



Can be displayed
as:



In a class diagram, a varying degree of detail of a class can be shown. Sometimes all that is shown is the class name. Alternatively the class name and the data are shown, or alternatively details of the data and actions are shown.

6.3.1 Views

At the beginning of this document it was stated that the identification and description of Classes/objects was partly a matter of individual interpretation. This is clear when one considers the examples given above. Now considering the NHS, a patient as perceived by a hospital chaplain is a very different object from that perceived by a director of finance. If you asked each to draw a Class diagram like those given on the previous pages, each would come up with a different set of data and activities.

- **A model is always context specific.**
- **It is never purely a reflection of reality - whatever that is!**

It is interesting to note that a large amount of effort in the modelling process is directed towards getting these differing views consolidated in some way. The potential problem is tackled in different ways depending upon the particular software development lifecycle chosen. Frequently a stakeholder or cultural/political analysis is carried out to see how important each of the views are, thus allowing a way to prioritise one view over another. Another method is used during the actual software development phase of the lifecycle where different views, or user interfaces, to the data can be developed for different user types. These are just two of a whole host of techniques available to tackle this problem.

This concept of a view is similar to that of a 'semantic purpose', which is described on page 22 of Rumbaugh (both editions same page).

The importance of being aware of potential problems with differing views in the healthcare sector is discussed in:

Willcocks P L. Mark A L. (1989) IT systems implementation: Research findings from the [health care] public sector. J I T [June] 92 – 103

The actual process that you go through to identify and develop the Classes you are already familiar with from the ERD document; as it is the same method that you used to identify Entity Types but obviously this time you need to consider the 'operations' side of things as well.

Exercise 4. Different viewpoints

Describe what might be the main differences in viewpoints between some of the following:

- ◆ community nurse
- ◆ GP
- ◆ clinical manager in a hospital
- ◆ anaesthetist
- ◆ day care coordinator
- ◆ paediatrician
- ◆ psychiatrist
- ◆ microbiologist
- ◆ director of finance in an acute trust
- ◆ patient

You may want to devise some type of table to help you organise your thoughts.

6.4 Activities/Operations/Methods

All the three above names in the title apply to the third section in each class in the class diagrams. They specify the behaviour of the class, its **dynamic aspects**. Clearly this aspect is very important, but discussion of it will be deferred to the document concerned with dynamic modelling.

7 Associations

When modelling a system, in terms of classes it is vital to be able to model the way in which they 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:

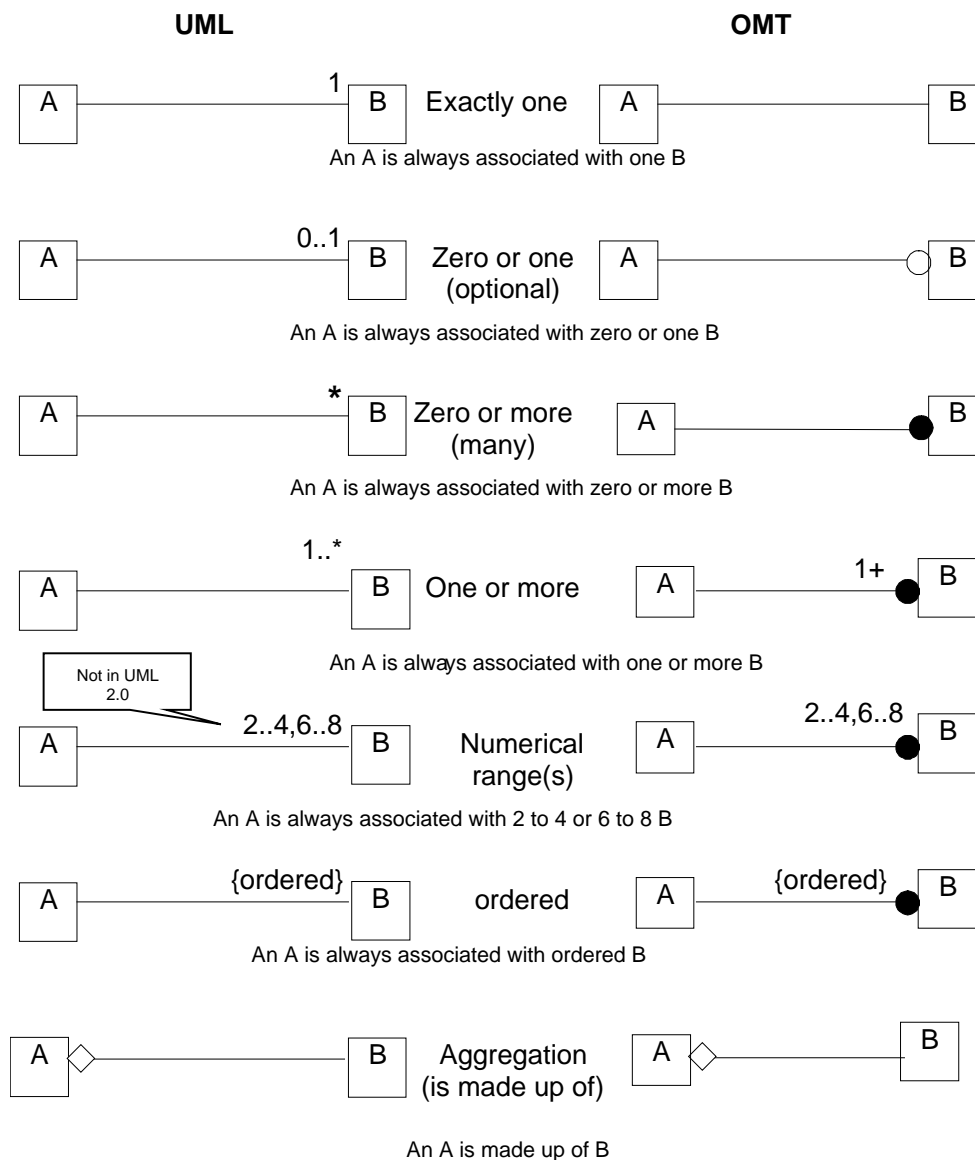
A person [class] *eats* bananas [class] or Sally [object] *eats* the banana [object] on her desk

The verb is often helpful to aid understanding of the association and is frequently used to provide a name for the association. If this is the case it is placed somewhere beside the association line on the diagram. At a further level of complexity, a **'role name'** can be placed at either or both ends of the association. A role name is simply taking the class and refining it in a particular way. For example: Person [class] works for NHS [class]. The association line at the 'Person' end could have 'employee' as the role name, with 'employer' as the role name at the NHS end.

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

Once again OMT/UML have sets of symbols to describe what are considered to be the common types of associations that exist between classes. These are given on the next page:

Types of Association Based on Fowler & Scott 1997 p59

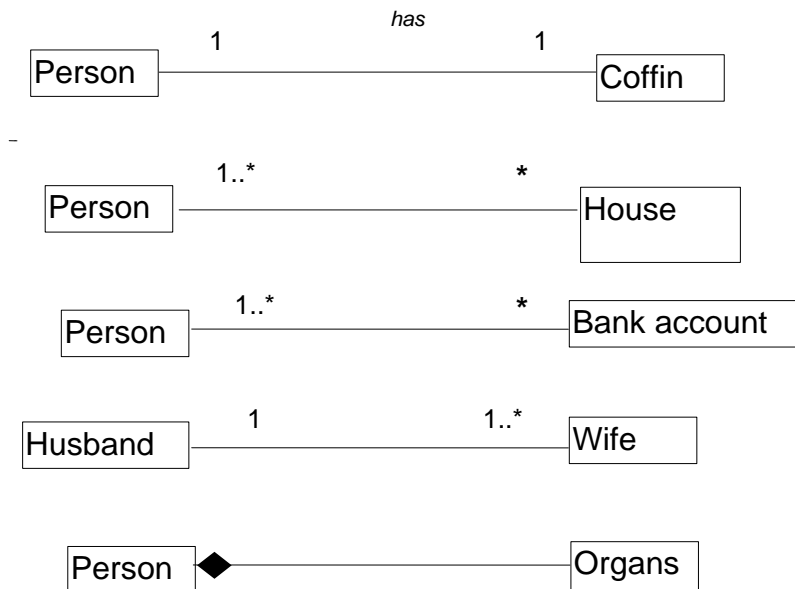


Associations are usually read from left to right. All of the above associations provide information about how many instances of one class can be associated with ('related to') another. Because these associations are concerned with the number of occurrences, they are often classified by their 'multiplicity'. 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') relations which are only subsequently elucidated.

7.1 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. Such an example might be a needle and syringe or a pan and lid, etc. Considering a slightly more complex situation, a frying pan may optionally have a lid, which would be represented by using the second type of symbol given below.



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 previous versions of UML in version 2 there seems to be no mention of the use of such discontinuous sets.

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.

You will have realised that the above types of associations are basically the same as those you came across in the ERD document, however the last one is new the diamond shape represents a special type of association called aggregation which is discussed next.

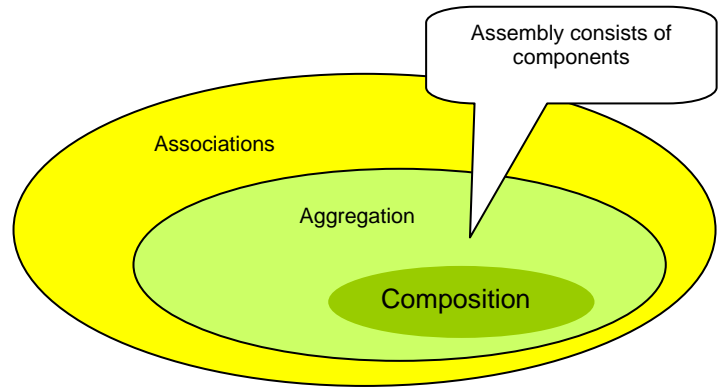
Exercise 5. Associations

Draw some associations between the classes you defined in the earlier tasks.

7.2 Aggregation and Composition ◆

These are special types of relationship. Let's consider aggregation first.

Aggregation can be translated into words as meaning **consists of**. In the last example in the diagram on the previous page, a person consisted of organs. In technical speech we would say that the person was the 'assembly' class and the organ was the 'component' class. The assembly class can be thought of as a type of collection. Taking another example, the MSc/DMI courses consist of a collection of units/modules.



It is important to realise that the assembly class may have additional properties that the sum of the component classes may not have. For example, considering the organ and body example each individual organ has a name but the body adds additional details when they are all combined such as occupation.

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

Composition is more clearly defined as it includes the constraint of co-existence of the part as part of the whole, quoting from the UML standard:

"A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive. Synonym: *composite aggregation*." UML 2.0 Infrastructure Specification, Terms and definitions section.

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-existent life cycles; the part can not exist without the 'whole'.

Table 7.3 - Graphic paths included in structure diagrams

PATH TYPE	NOTATION	
Aggregation		
Association	 	from uml specification 2.2 page 141
Composition		

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, shown on the previous page. 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 the other is filled.

Exercise 6. Aggregation and Composition

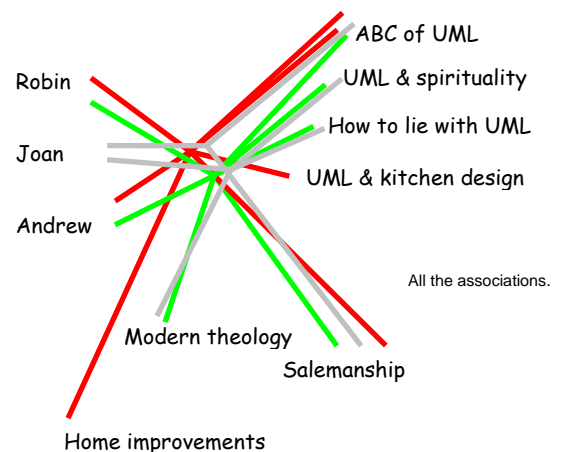
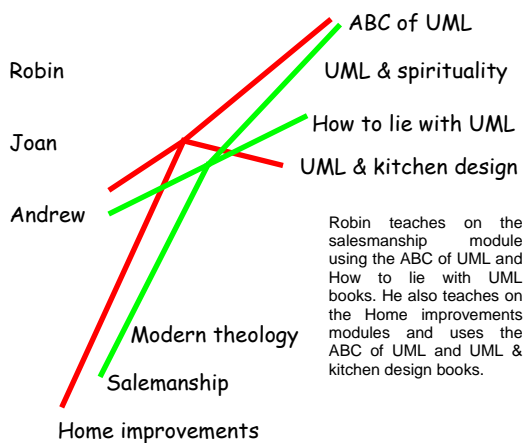
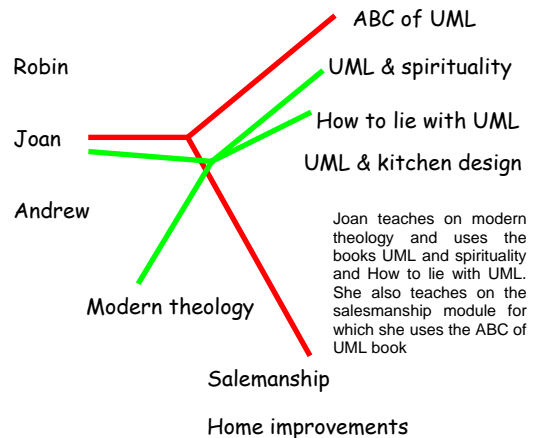
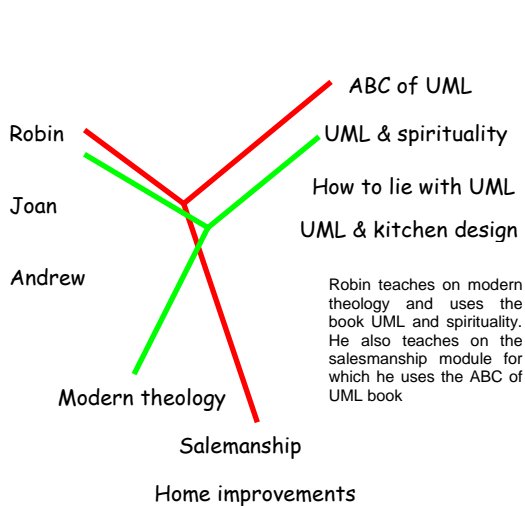
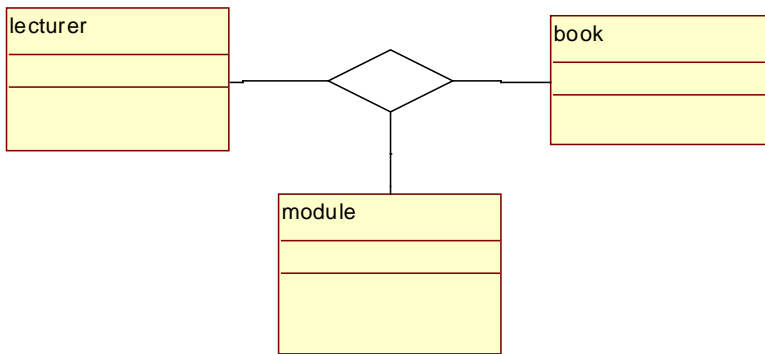
Draw class diagrams showing the data for each of the following aggregations / compositions: bed, ward and hospital, canteen department and a department of your own choice.

7.3 Ternary associations

So far I have only mentioned associations which involve 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'.

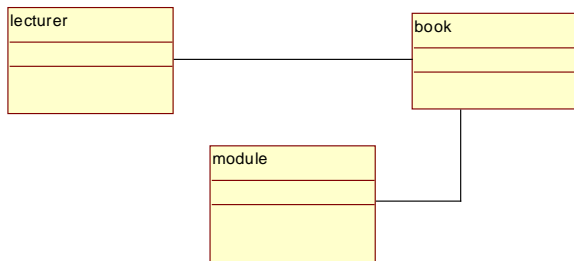
In other words we are defining a set of three specific values (L,B,M) in contrast to two specific values for an association involving two classes.



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?

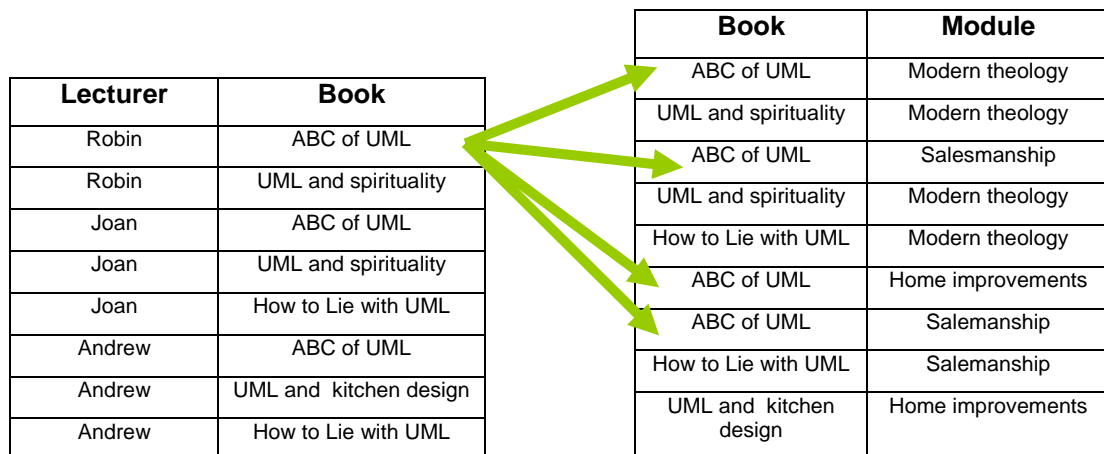
7.3.1 Breaking Ternary associations into Binary ones

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 remodel 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?

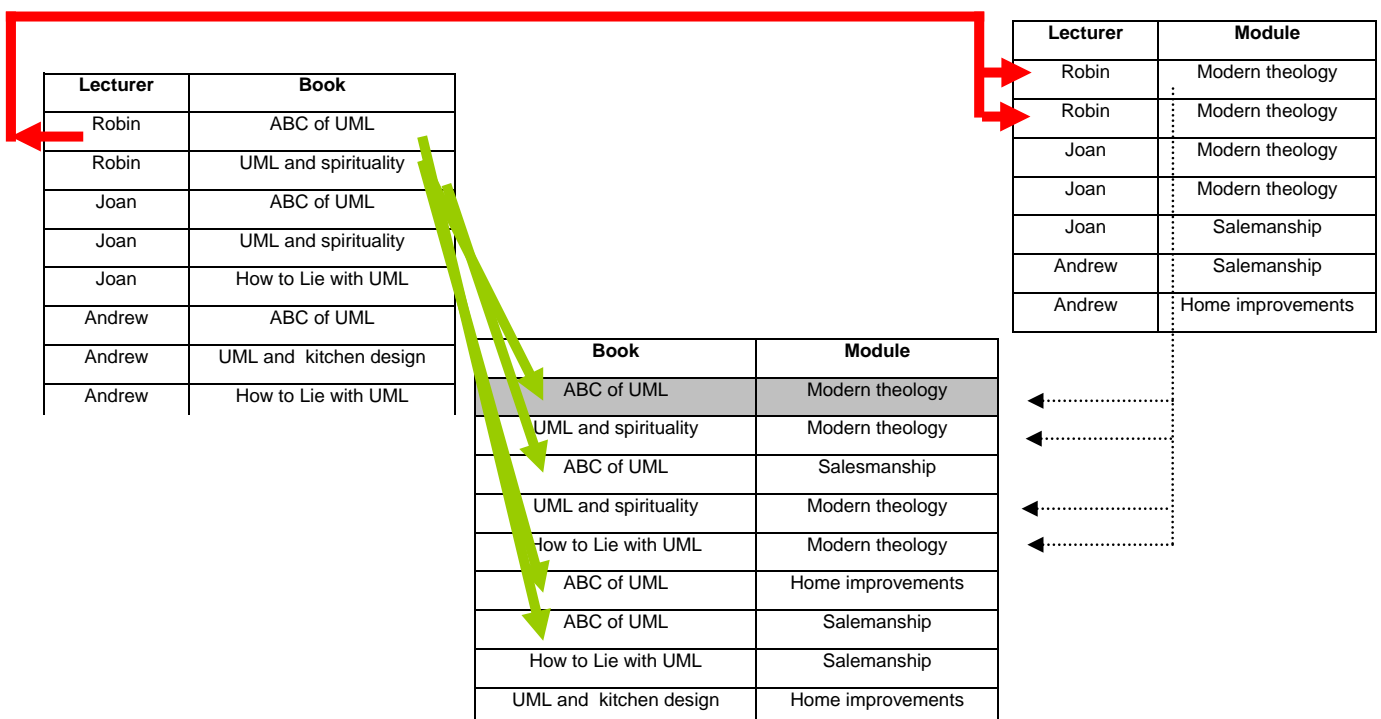


Also more worryingly by following through the new associations from Book to module, we do not really know which module which lecturer teaches on. Take for example, lecturer instance robin who is associated

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!



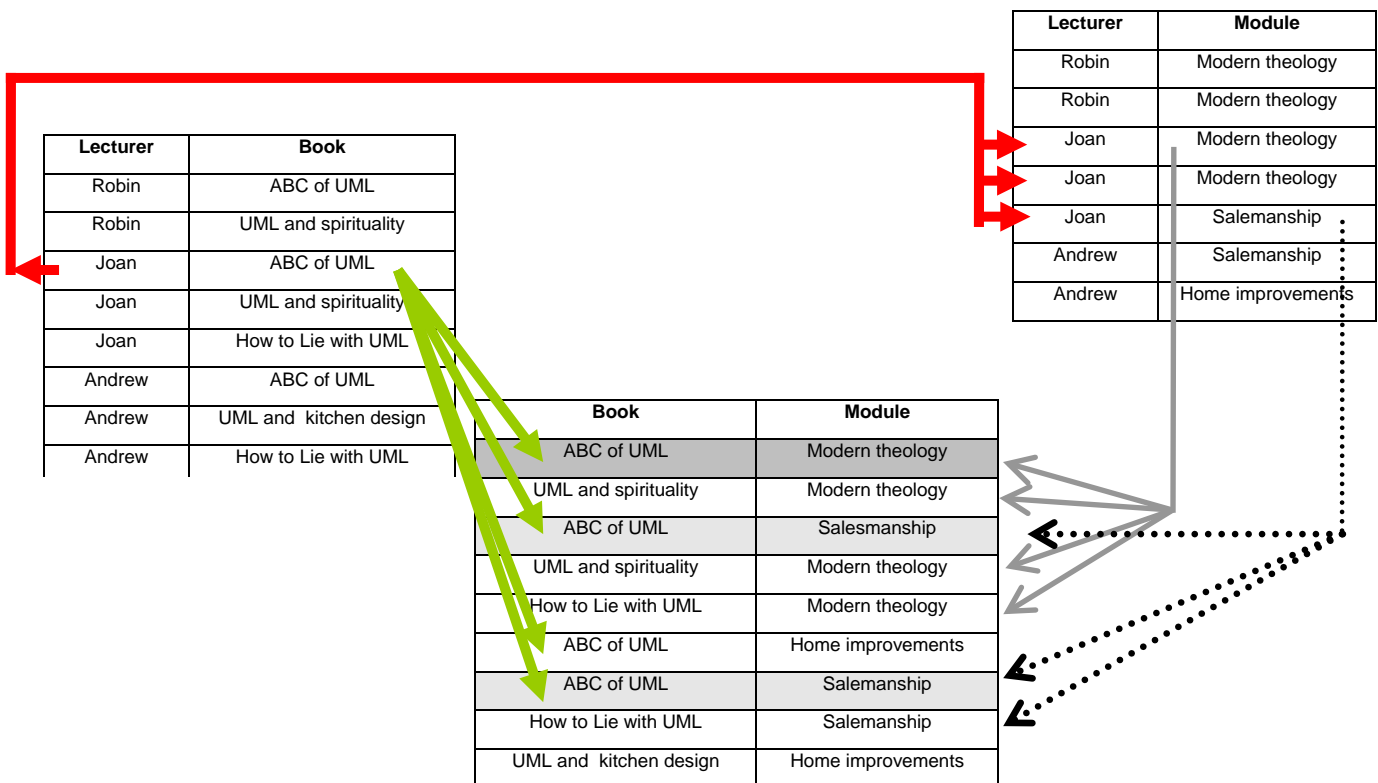
Can we mimic the Ternary association by adding another binary relationship between Module and Lecturer?



Class modelling with UML

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?

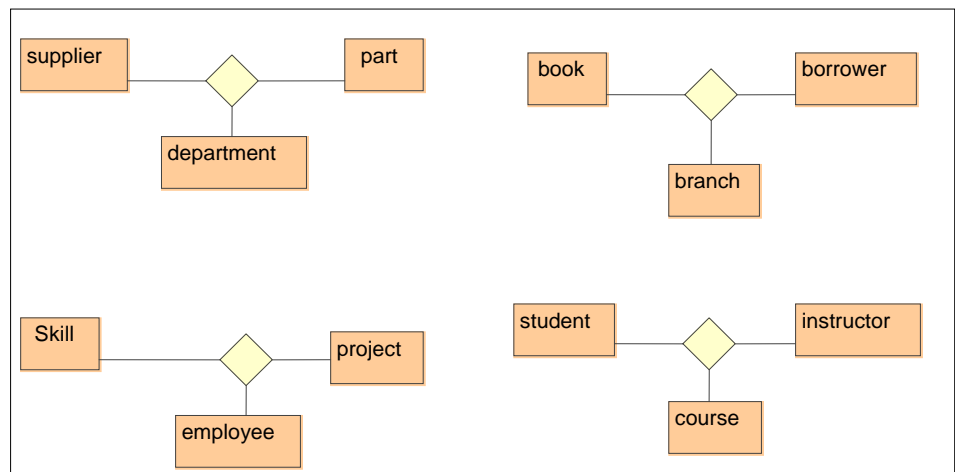
Taking another example, that of a instance of Lecturer, Joan lets see if we can identify a unique instance of book she uses for a particular instance of module, put simply can we identify which book(s) she uses for a particular module. From the diagram below it looks that this time we can't because there 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.



7.4 Recursive associations

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

So far we have discussed examples of relationships between classes; however, it is possible to an association within a class. Such a relationship is called recursive.

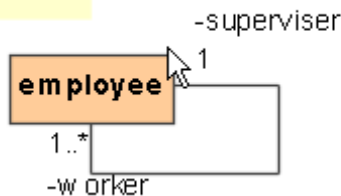


The usual example given is that of an EMPLOYEE entity type that has a relationship with itself called supervisor. In other words an EMPLOYEE instance can relate to another EMPLOYEE instance in a supervisor role.

Clearly a 'one to many' or 'many to many' recursive relationship may also exist. (see Carter 1995 p61 - 68, Elmasri & Navathe 1989 p. 49) Some writers suggest that these recursive relationships are acceptable in early models but should be converted to 'one to many' or 'one to one' relationships, as the model is refined. For example the above recursive relationship could be modelled by introducing a new entity type called ROLE. This was discussed in the ERD document.

7.4.1 Association End Names

end name to specify role



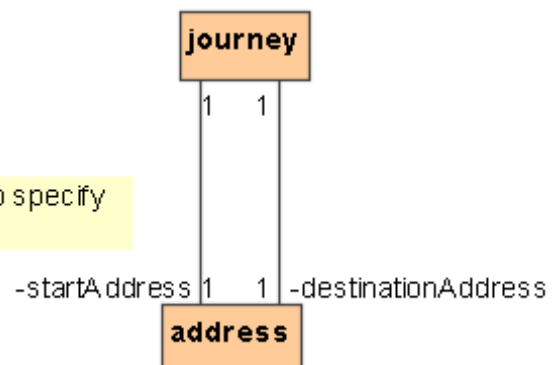
In UML the concept of Association end names is used to specify a role a particular instance may play in the association. Considering the above example an Employee may only have one supervisor and the supervisor will manage one or more employees.

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

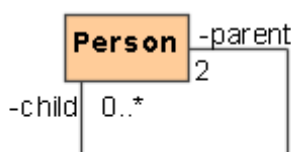
using association end names as shown in the diagram below (adapted from p. 97 Bennett, Skelton & Lunn 2005).

In contrast to the suggestion above concerning the use of 'Role' classes Blaha & Rumbaugh 2005 p. 32 suggests that recursive associations should be retained using association end names and provides the following example.

end name to specify role



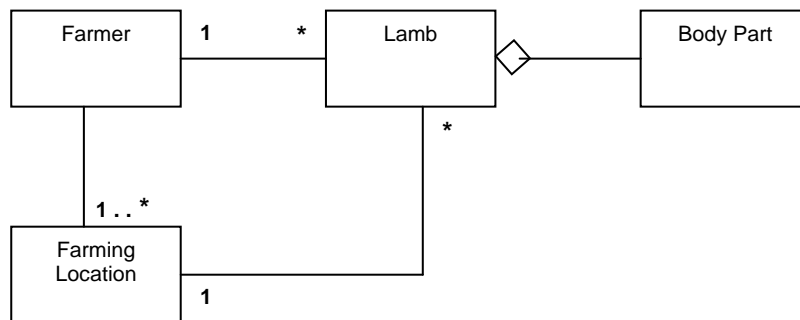
Incorrect model



correct model

Exercise 7. 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. A farmer has one or more lambs.
 - b. A farmer has one lamb.
 - c. A farmer has zero or more lambs.
 - d. A lamb does not necessarily have a farmer associated with it.
 - e. 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. A farmer has one or more farming locations.
 - b. A farming location is associated with zero or more lambs.
 - c. A lamb is always associated with one farmer.
 - d. A lamb can be associated with more than one farming location.
 - e. A lamb consists of body parts.

 3. The above diagram uses:
 - a. OMT notation
 - b. UML notation
 - c. ERD notation
 - d. No formal notation
 - e. Chen notation
-

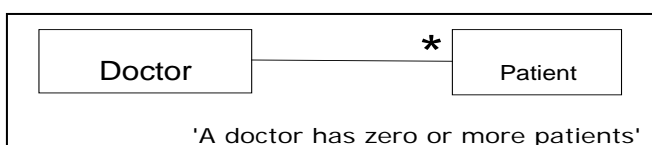
8 Mapping Associations to Databases

The object 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 next sections are examples of the relationship between associations in object models and relationships in relational DBMSs.

8.1 One to Many Relationship

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:

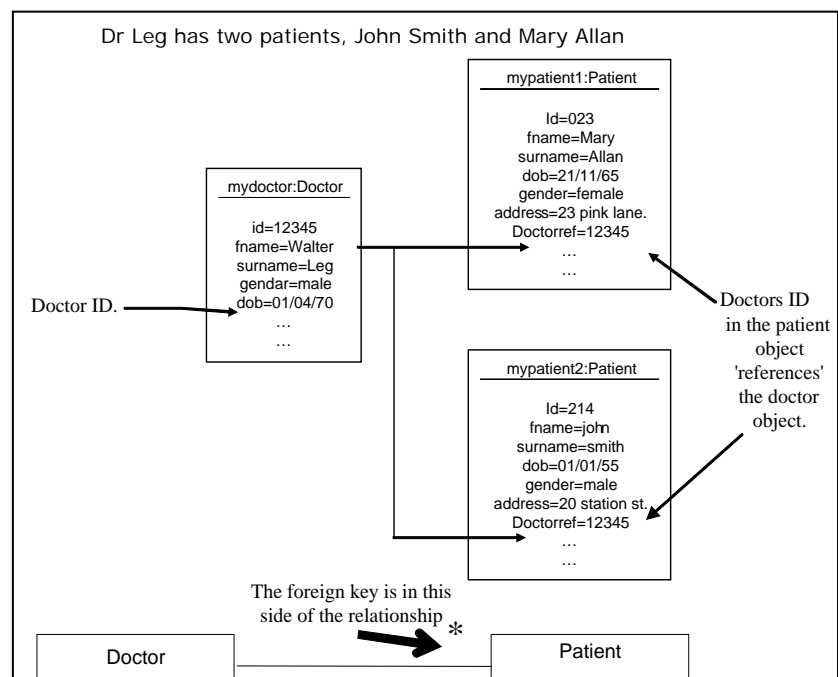


But how do we turn this into something that can be represented as one or more tables (i.e. something that can be used by a computer)? The

answer is that we make use of 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>

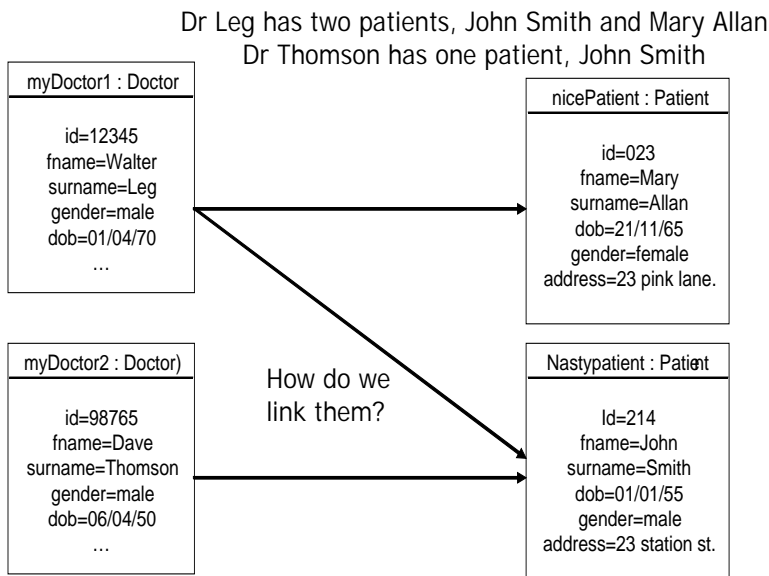
A foreign key is a primary key in one table that is embedded in another (or the same) table. In the example given below, doctor ID is the foreign key value in the patient object. It allows the implementation in a DBMS of the association shown in the class diagram above. The diagram below shows exactly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



Exercise 8. Converting Classes/objects to tables/records

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

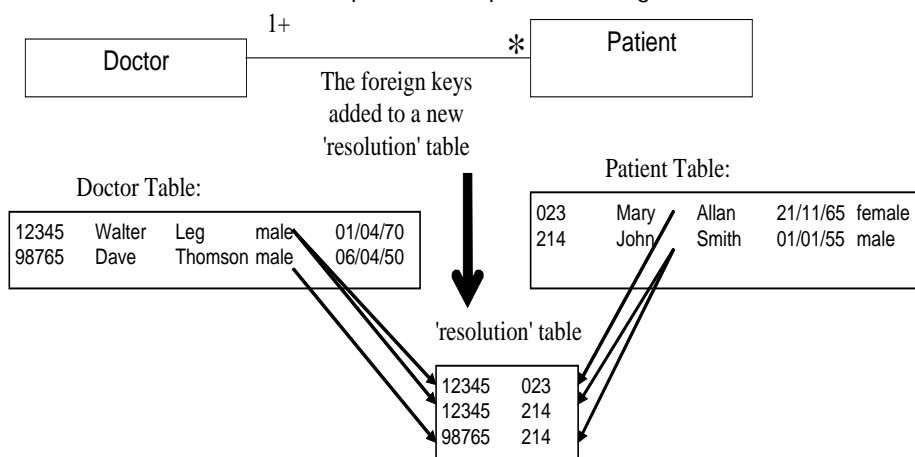
8.2 Many to Many Relationship



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 object 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.

A Dr has zero or more patients. A patient belongs to one or more Drs.



Exercise 9. MCQs

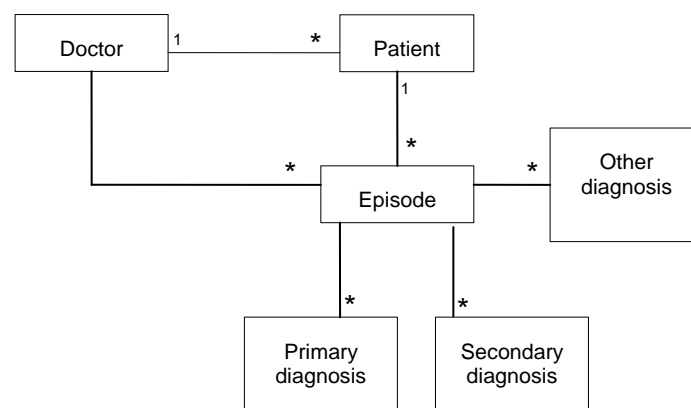
- When mapping a class to a database, what does it become?
 - A record
 - A database schema
 - An ERD
 - A table definition
 - An individual data item
- When mapping an instance of a class (object), what does it become?
 - A record
 - A database schema
 - An ERD
 - A table definition
 - An individual data item

Exercise 10. Identifying foreign keys and associations

This exercise requires you to consider the models below and indicate in which class the foreign key(s) will need to be added for implementation in a relational DBMS. **This is not part of UML.**

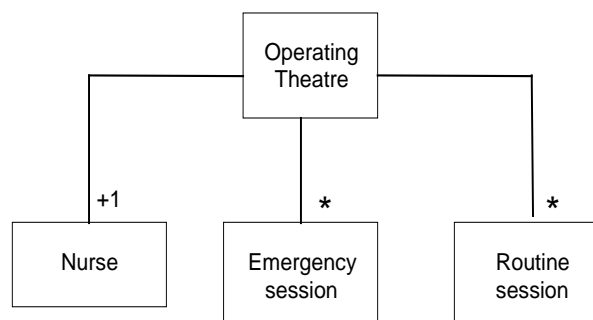
1. General Practitioner (GP) consultation model¹:

- A doctor has zero or more patients
- A patient can only belong to one doctor
- A patient can have zero or more episodes
- A episode is related to one patient
- A episode has zero or more primary, secondary and other diagnoses associated with it
- A diagnosis is only related to one episode.
- A doctor has zero or more episodes
- A episode is related to a single doctor



2. The hospital model part A:

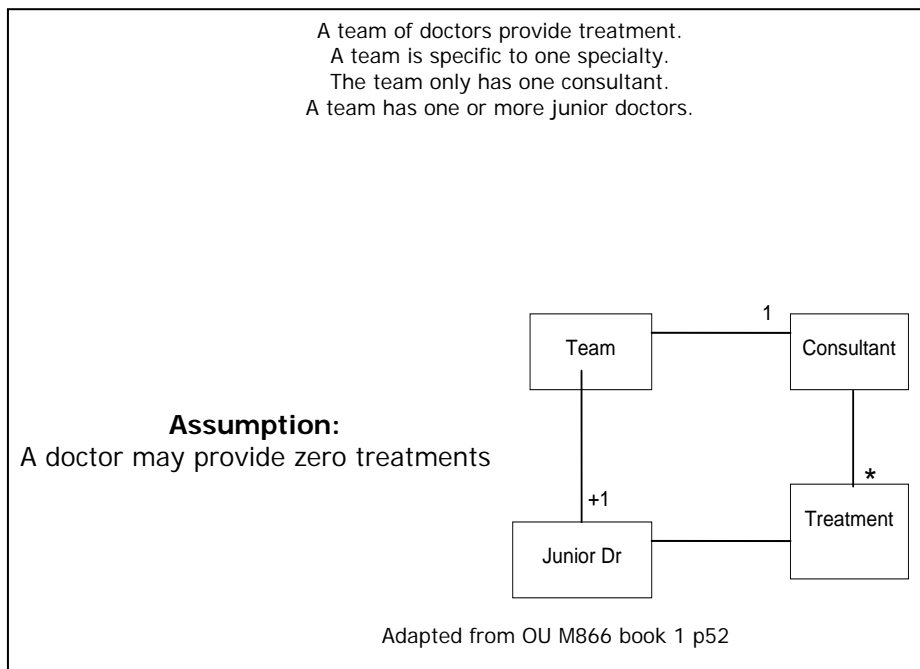
A operating theatre has zero or more sessions allocated to it which are either routine or emergency.
A theatre has one or more nurses allocated to it.



Adapted from OU M866 book 1 p52

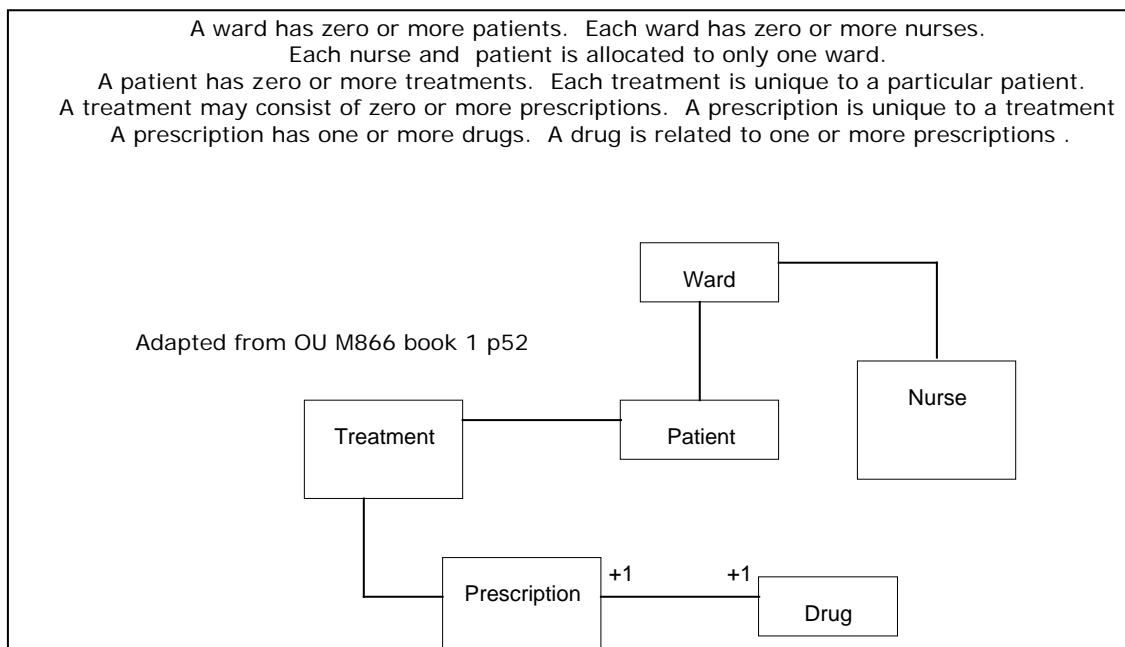
¹ Answer: In patient object: Dr ID. In Episode object: Dr ID and Patient ID. In each of the diagnosis tables: episode ID.

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):

This example requires you to first pf all annotate the uml diagram to identify the type of association, from the narrative description before indicating where the foreign keys will be placed.



The above exercise concerning foreign keys was to get you to understand the concept. You would NOT normally display foreign keys in a UML class diagram. And therefore should not usually do this.

9 The Meaning of Relationships (Semantics)

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 object 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).

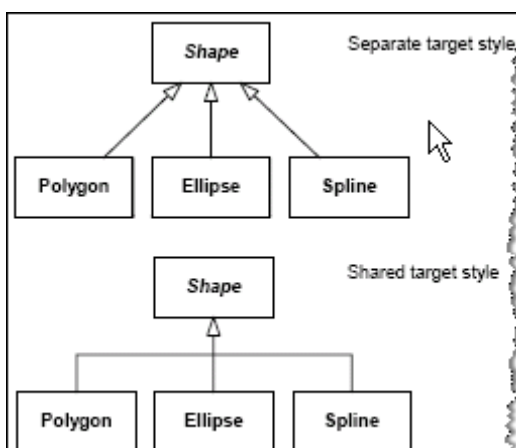
The term **Ontologies** has been used by the Artificial Intelligence (AI) community to describe these more complex models, which also include a controlled vocabulary. You can find a good introduction to the subject in Mustafa Jarrar's PhD thesis at his web site <http://www.starlab.vub.ac.be/staff/mustafa/> (unfortunately he seems to have recently removed the content and left the table of contents only); his article *Data modelling versus Ontology engineering* provides an alternate source at [http://www.starlab.vub.ac.be/staff/mustafa/publications/\[JDM03\]LNCS03_V1-11.pdf](http://www.starlab.vub.ac.be/staff/mustafa/publications/[JDM03]LNCS03_V1-11.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.

It is possible to derive the models we have been using 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 object model in each of the object diagrams so far.

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.htm>

10 Inheritance



Inheritance is a very powerful tool for organising what often appear to be very complex situations into something that is more manageable. When modelling a situation, this is one of the later actions that takes place after all the classes and associations have been quite thoroughly described. This process itself involves the development of several drafts.

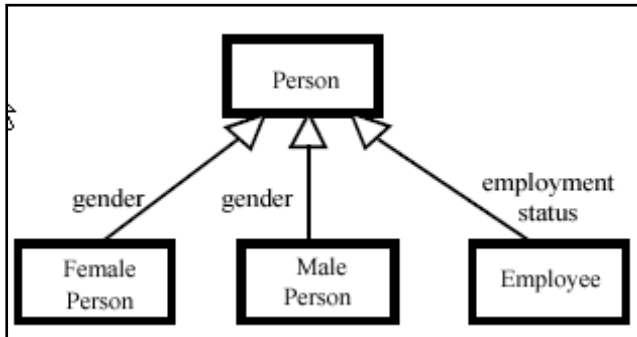
Inheritance is the process of inheriting data and/or activities from a parent (**superclass**) to that of a child class (**subclass**). An example would be the subclass male person who would inherit all the detail in the human superclass but add some of his/her own. Often the parent 'superclass' is something that exists in the model but not in the real world, although this does not have to be the case as in this example. When an instance of the parent class does not exist, it is known as an **abstract class**.

Generalisation is just describing inheritance from the opposite perspective. The subclass is a **specialization** of the superclass and therefore the superclass is a generalisation of several subclasses.

In a UML Class diagram you show inheritance by using either a single triangle or one for each (generalisation) line, but this can lead to confusion (see below for why this is the case).

10.1 Generalisation Sets

From the above diagrams concerning various shapes it is possible to see that while it is helpful to show various generalisations it could also be confusing, Take for example:



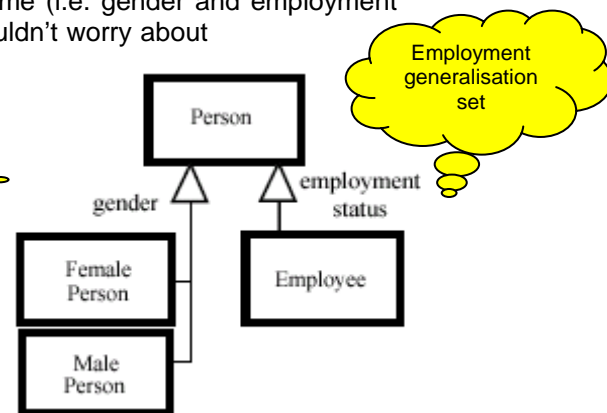
Does this mean that an employee can not have male or female specific attributes / operations? Unfortunately it is not clear from the diagram that what we want to describe is that a person is either male or female and also has an employment status.

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). In the UML this is called the power type classifier, but I wouldn't worry about that too much.

However this can still be confusing for within a particular Generalisation set we might want to specify that an instance of person might possess characteristics of more than one subtype. This can be achieved by using Constraints.

Gender generalisation set



10.2 Constraining Subclasses

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

First two mutually exclusive options:

- **Complete** – All children (i.e. subclasses) have been specified (whether or not shown). 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.

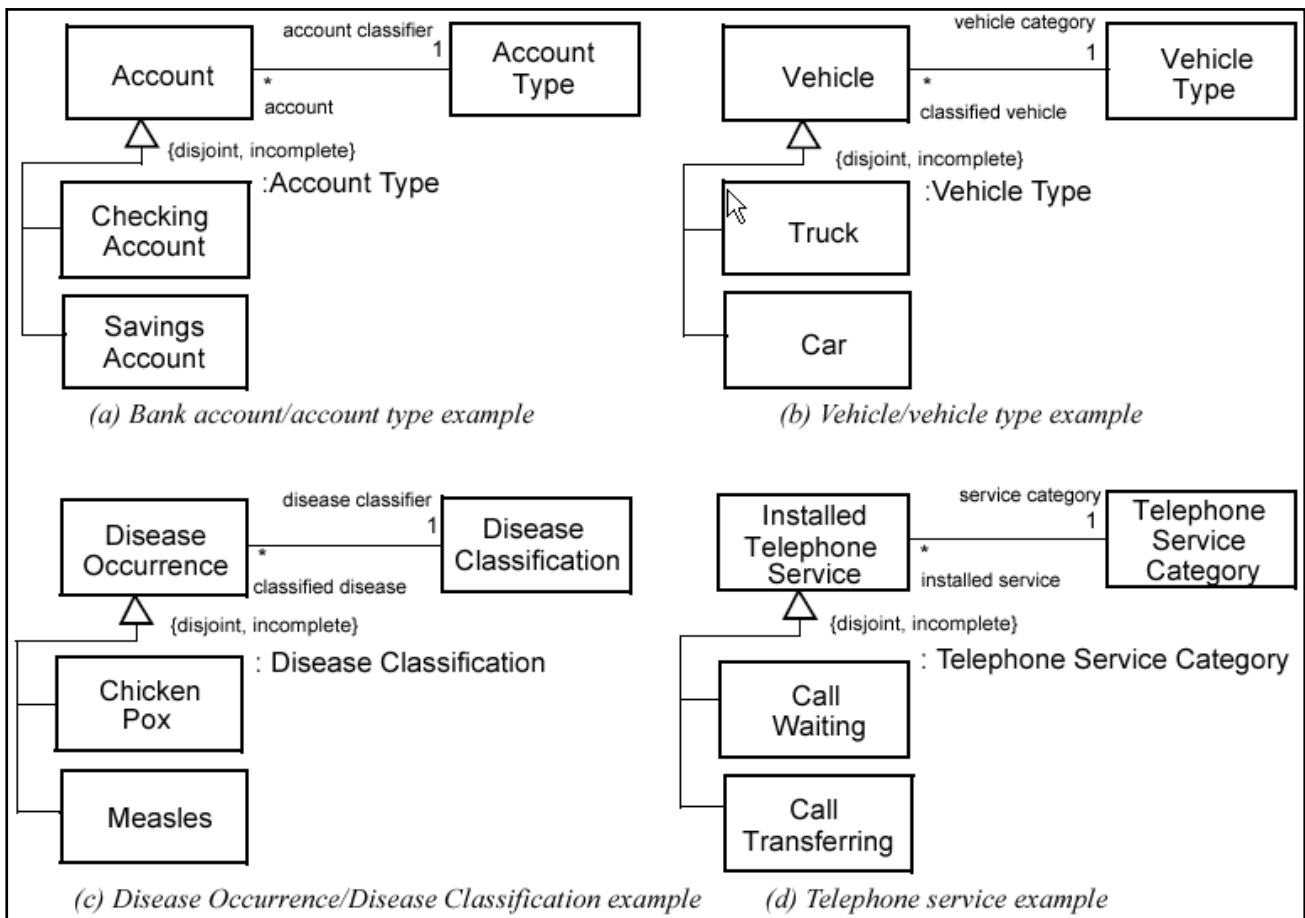
Second two mutually exclusive options:

- **Disjoint** – A class only inherits from one subclass within each Generalisation relationship.
- **Overlapping** – A class can inherit from several subclasses within each Generalisation relationship.

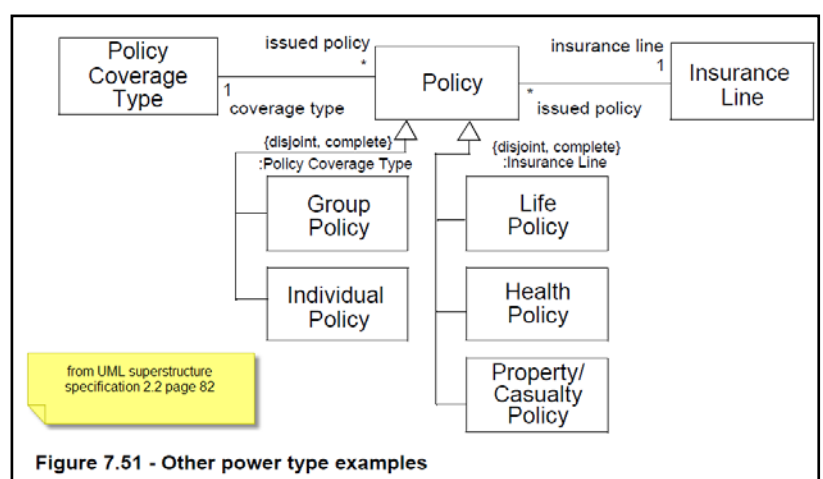
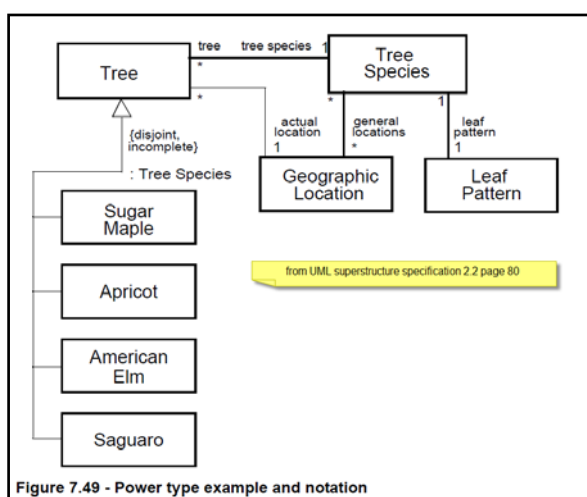
Let's consider an example given in the UML 2.0 superstructure (p76). Person could have two Generalisation relationships: Manager or Staff. This would be disjoint because every instance of Person must either be a Manager or Staff. In contrast, Person could have two Generalisation relationships: Sales Person and Manager. This Generalisation set would not be disjoint because there are instances of Person which can be a Sales Person and a Manager.

If no constraint is shown on the generalisation line, the Default is {incomplete, disjoint} (p.77 uml 2.0 superstructure).

Here are some more examples from the UML 2.0 Superstructure specification document:

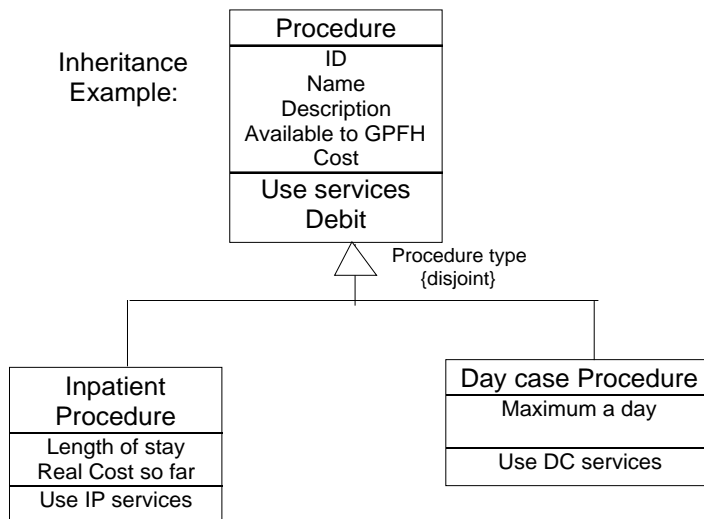


And some more:



We end this section by considering a hospital example.

10.3 Inheritance - Hospital Example



The diagram below 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 (objects) 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 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.

Let's recap:

The hollow triangle in the above diagram indicates that both inpatient and day case procedures are inherited 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**. OMT used a different symbol for this: a filled in triangle; UML does not.

The text beside the triangle 'procedure type' is known as a **discriminator** and is an attribute (data) whose value differentiates between the subclasses. Often this data item is contained in the superclass.

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 subclass called 'other' as well.

Exercise 11. Generalisation/Inheritance

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

- ◆ Cake mix as superclass; define some subclasses
- ◆ Employee payment as a superclass; define some subclasses
- ◆ NHS provider unit as a superclass; define some subclasses
- ◆ A superclass of your own choice; define some subclasses

11 Encapsulation

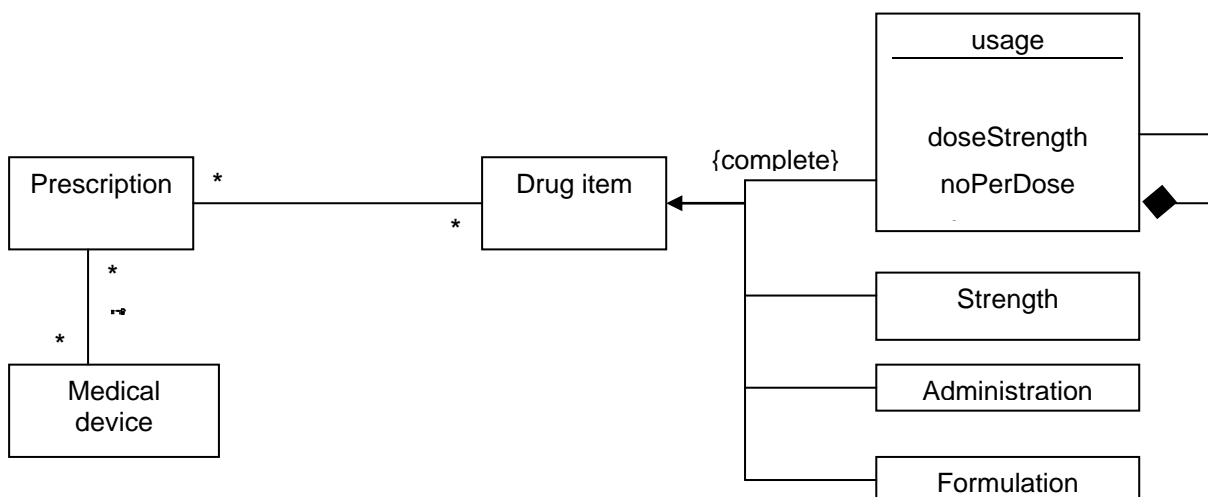
Encapsulation has two different but related meanings. Firstly, it is used to mean the idea of an object/class, in the manner we have been describing it above. That is, something that has both data and actions (Blair, 1989 quoted in Appleby 1991; Programming language concepts and paradigms). It can also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


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

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

Exercise 12. MCQs

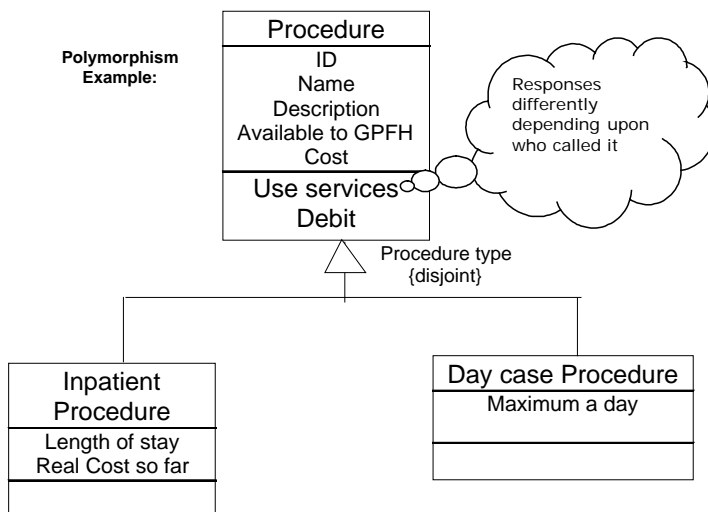
The following class diagram is of a prescription class. Formulation is to do with how the drug is prepared (e.g. tablet, paste etc). Administration route is the method of taking the drug (e.g. oral etc), and usage details are basically instructions for the patient besides administration route.



- In the above diagram, which is the incorrect statement?
 - A prescription has zero or more drug items.
 - A medical device is related to zero or more prescriptions.
 - Each drug item consists of usage, strength, administration route and formulation details.
 - Each drug item can contain a single usage detail.
 - A single drug item can be related to zero or more prescriptions.
- In the above diagram, what does the symbol  indicate?
 - Usage details consist of one or more such sets of details.
 - Usage details consist of zero or more such sets of details.
 - Usage details are related to several other usage details.
 - Usage details are not clearly defined.
 - The relationship is ambiguous.
- The "{complete}" indication means the following:
 - A drug item must contain details of one or more of the following: usage, strength, administration and formulation details.
 - A drug item must contain details of usage, strength, administration and formulation.
 - A drug item is only 'complete' when it contains information about the frequency details.
 - A drug item is only 'complete' when it contains information of usage, strength, administration and formulation.
 - A drug item is also called a 'complete' item.

12 Polymorphism

This is another aspect of the 'object' paradigm that is not so important during the modelling stage but more important during the software development stage.



Polymorphism means that activities with the same name can behave differently in different classes. For example, consider the above diagram of various procedure types. The inpatient procedure class has an activity 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 objects taking care of what actually happened. The advantage of this would be that the hospital manager object would then only need to send one message to both objects 'use services' rather than a different one to each.

13 Larger models and using Case tools

We have now covered the main characteristics of Object Oriented Modelling. However, there are two additional concepts which are important in UML which need consideration: Packages and Stereotypes. Both of these concepts become more important in larger models or when one uses a case tool such as System Architect.

13.1 Packages

A Package in UML is just a container. 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.

The System Architect case tool has a good tutorial which involves modelling a hotel business, where you divide the model into three separate packages: Reservations, Human Resources and Building Maintenance.

In ERD modelling in System Architect they talk about 'subject areas' (see the SA advanced ERD tutorial) 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.

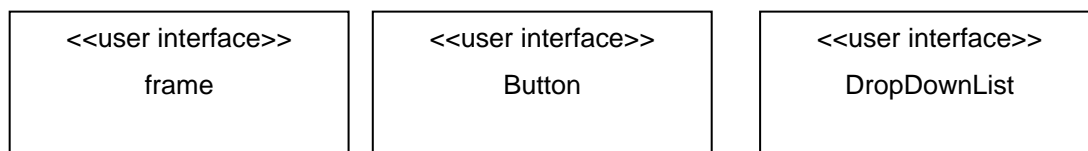
People often get unduly nervous about packages. Just think of them as folders for keeping similar UML elements; after all, the visual icon does look like a Microsoft folder icon.

13.2 Stereotypes

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

Again this is a type of UML element which allows you to group others together. In this instance a stereotype is a special type of class which you cannot use on its own but which you can use as a basis for classes that you create in your models. Each class that makes use of the Stereotype indicates this by including the name of the stereotype above its name.

Three cases that make use of a stereotype called 'user_interface' (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. Fowler 2004, who always offers sensible advice, hints at this (p66) as well.

What you think is a stereotype is usually just an abstract superclass. For example the class Employee might be used in a model of an institution showing all the types of employees, but will be represented as a particular subtype such as full-time employee or student etc.

Exercise 13. Developing packages

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

Suggest what packages you might consider for one or two of the scenarios in the scenarios document

You can find the document at: <http://www.robin-beaumont.co.uk/virtualclassroom/contents.htm> and follow the links.

14 Exercises

You have now covered the most important aspects of Object Oriented Modelling, particularly that concerned with looking at the 'What' rather than 'How' aspect. To help you consolidate your knowledge I have provided a few exercises below.

Exercise 14. 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 15. 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

Using the knowledge you gained from the document "An Introduction to Entity Relationship Diagrams (ERDs)" define and develop a possible set of Classes and associations.

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 16. 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 object 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 understand of the model(s).

You might want to use a group working evaluation sheet.

Optional Exercise 17. 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.)

15 Summary

After all the information and exercises that you should have done to get this far, you must feel quite exhausted! Let's recap what has been covered:

- Description of the key characteristics of object oriented modelling
- Details of Class diagrams
- Experience of creating Class diagrams from narratives
- Experience of indicating on Class diagrams the necessary foreign keys to implement them in a database
- Discussion of the differences between UML and OMT Class diagrams
- Discussion of Generalisation sets and how useful the concept is in developing Class diagrams
- Introduction to Packages and Stereotypes

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 other 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>

16 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.

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

Blaaha M, Rumbaugh J, 2005 Object-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 object 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 Object-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 Object-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.

17 Web Links

An excellent glossary of UML and object modelling terms can be found at:

<http://www.csci.csusb.edu/dick/samples/uml.glossary.html> The relative link also contains a good example of a teacher/student/course class model at <http://www.csci.csusb.edu/cs202/uml1.html>

Links to most UML sources including all reference documents including most current as well as introductory tutorials:

<http://www.rational.com/uml/resources/documentation/index.jtmpl>

Quick reference guide to UML:

<http://www.rational.com/uml/resources/quick/index.jtmpl>

Object Modelling group - International body responsible for setting OO standards:

<http://www.omg.org>

Alistair Cockburn's stuff on use cases:

<http://alistair.cockburn.us/crystal/articles/sucwg/structuringucswithgoals.htm>

The UML bibliography:

<http://www.db.informatik.uni-bremen.de/umlbib/>

Past student's recommendations:

A quick introduction to UML:

<http://bdn.borland.com/article/0,1410,31863,00.html#use-case-diagram>

Class and sequence diagram guidelines and more:

<http://www.agilemodeling.com/style/classDiagram.htm>

<http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

All above links were active on 18/03/2007.