Ch 2 – Modelling the Real World

Introduction

Many of today's problems in data management can be traced back to how the data has been modelled (i.e plans for input/output & storage on disk). Most implementations fail to follow established techniques for relational data modelling and programming. Projects often do a very poor job at the basics: naming conventions, establishing data definitions and defining relations. This then leads to problems around integrity and accuracy of the data/information that are difficult to resolve.

OOP & OOD is an attempt to model more realistic and manageable data and programs.

Object-oriented programming (OOP) is a based around "objects" and their interactions to design applications and computer programs. It is based on several techniques, including encapsulation, modularity, polymorphism, and inheritance. It was not commonly used in mainstream software application development until the early 1990s. Many modern programming languages now support OOP.

Object oriented programming roots reach all the way back to the 1960s. As hardware and software became increasingly complex, researchers studied how software quality could be attained and maintained. Object-oriented programming was deployed in part as an attempt to address this problem by strongly emphasizing discrete units of programming logic and re-usability in software.

The Class

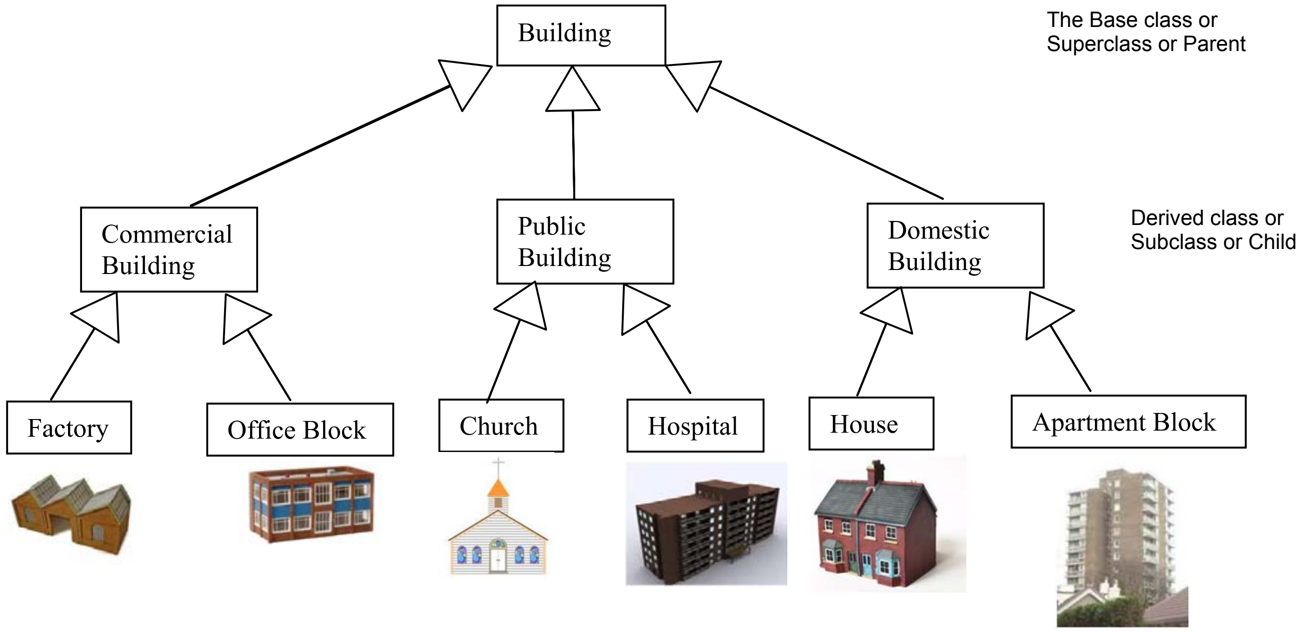
A class is a blueprint or prototype from which objects are created. In the real world, you'll often find many individual objects all of the same **kind**. There may be thousands of bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

Inheritance

Once a class has been defined it can be used in the construction of another class. Take the bicycle example. Separate classes are defined for wheel, frame, handle bars, saddle, etc. A particular class of bicycle (say Mountain bike) inherits from each of these classes to form its own class.

Inheritance is a very powerful feature of OOP. By organising classes into a 'classification hierarchy', it gives an extra dimension to the encapsulation of abstract data types. It enables classes (and therefore objects) to inherit attributes and methods from other classes. The inheriting class can then add extra attributes and/or methods of its own.

This example hierarchy shows how classes of buildings inherit from a base (or basic) class.
`Commercial', 'Public' and 'Domestic' buildings are therefore all **'a kind of'** building.
'Kinds of' commercial building might be factories, office blocks, hotels etc,
'Kinds of' public buildings hospitals, churches, libraries, railway stations and so on.
Apartment blocks and houses are `kinds of' domestic buildings



The use of arrows indicates an 'inherits from' relationship (i.e, derived classes point to their base classes), a notation used in the Unified Modelling Language (UML).

Exercise 2.1

1) Trees

Draw a classification hierarchy with the class 'Tree' at the top. Justify your distinction between types, and suggest attributes which might be inherited by derived classes. Suggest a class which might be 'a part of' one of your classes (and therefore not part of the hierarchy.

An example might have 'tree' being the base class for 'deciduous' and 'evergreen , with perhaps 'fruit bearing' as a further subclass of 'deciduous'. A simple attribute might be 'height'. 'Branch would be an example of 'a part of' a tree, rather than 'a kind of' tree.

2) Animals

Draw a classification hierarchy with the class 'Animal' at the top. Justify your distinction between types, and suggest attributes which might be inherited by derived classes. Suggest a class which might be 'a part of' one of your classes (and therefore not part of the hierarchy.

An example might have 'Animal' being the base class for 'Mammal' and Bird , with perhaps 'cow', 'pig' 'horse' as a further subclass of 'Mammal'.

Unified Modeling Language(UML)

Unified Modeling Language(UML) is a special notation (mostly graphical) which is used to specify, visualize, and document an architecture of software system.
This includes its structure and design, in a way equally understandable for all people involved in the software project.

UML actually defines twelve types of diagrams, divided into three categories:

- ***Structural Diagrams***
  Class, Object, Component, Deployment.
- ***Behaviour Diagrams***
  Use Case, Sequence, Activity, Collaboration, Statechart
- ***Model Management Diagrams***
  Packages, Subsystems, Models

Software Systems must be structured in a way that enables scalability, security, and robust execution under stressful conditions, and their structure (often referred to as their ***architecture***) must be defined clearly enough that maintenance programmers can effectively modify and upgrade software after the original authors have moved on to other projects.

A well-designed architecture has many benefits:

-dealing with complexity and a team-work approach.
-effective modification and upgrade of software
-enables **code reuse & a *library of components*** each component representing an implementation stored in a library of code modules ready for when another application needs the same functionality.

UML meets all of the these requirements and is a commonly accepted notation.

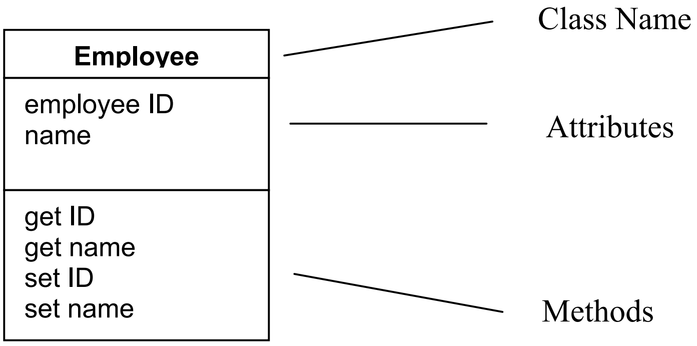
Advantages of Modelling

Modelling is the designing of software applications before coding and is an Essential Part of large software projects (medium and even small projects as well). Modeling can assure that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code renders changes difficult and expensive to make.

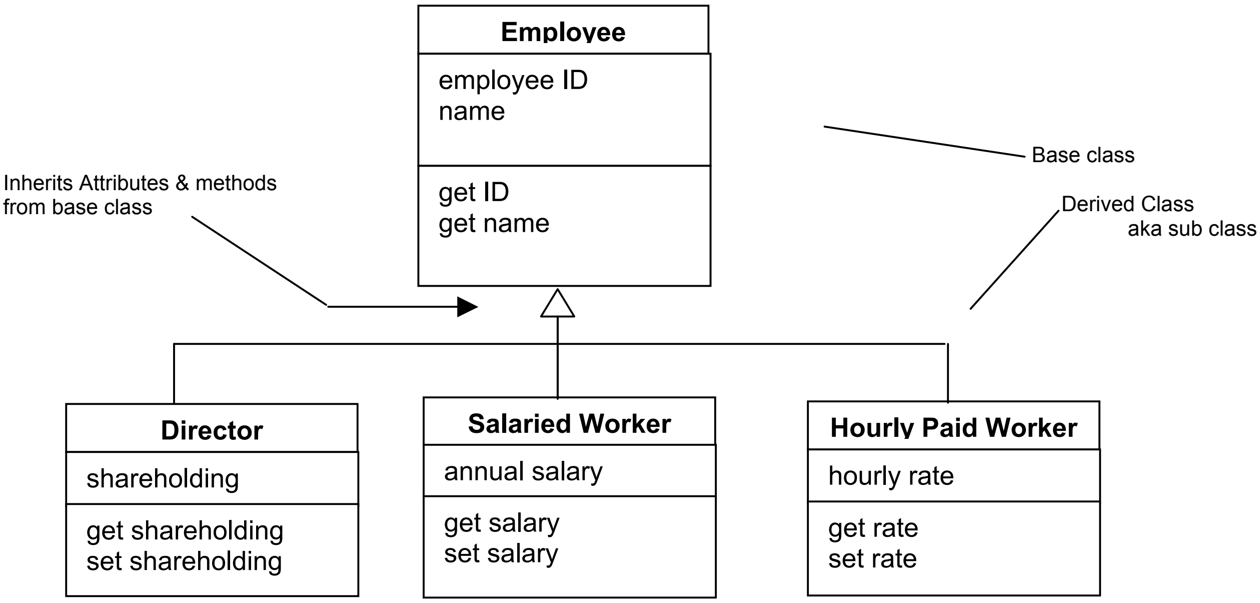
Structural

Class Diagram

A ***class*** is represented in UML as follows:



UML Can also shows how a class can inherit Attributes & methods from a "base class"



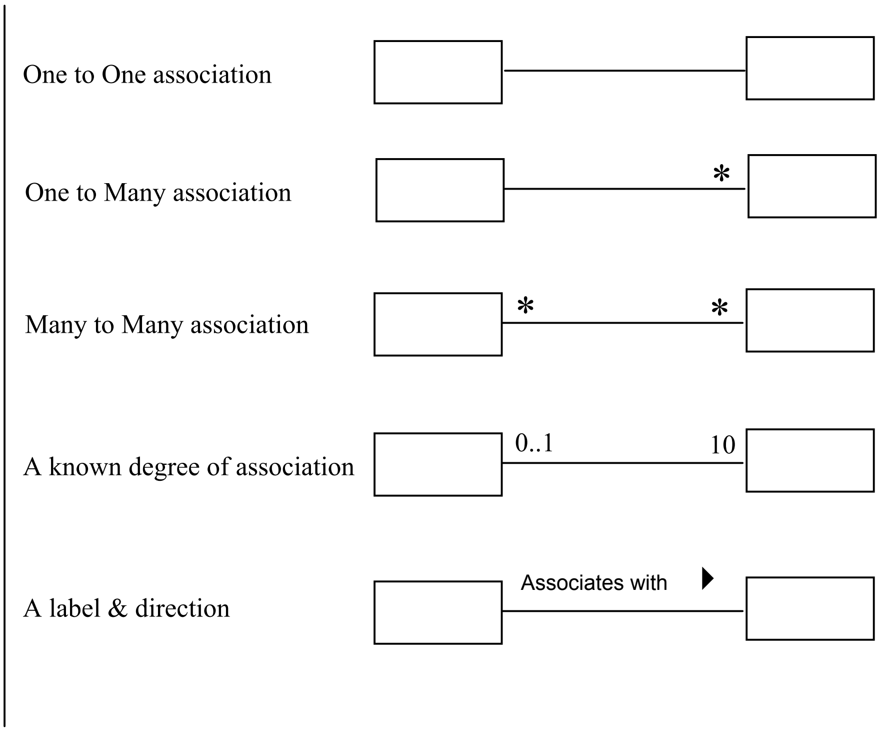
Inheritance is represented by an arrow and shows Salaried Worker is a **derived** or **subclass** of Employee, inheriting all of the Attributes and Methods of the Employee Class.

Exercise 2.2

A bank has customers; and a customers may have a current account and a savings account. Draw a class diagram showing the classes (class name, attributes & methods) for the bank.

Association

When objects are linked in some way there is an **Association** between them.

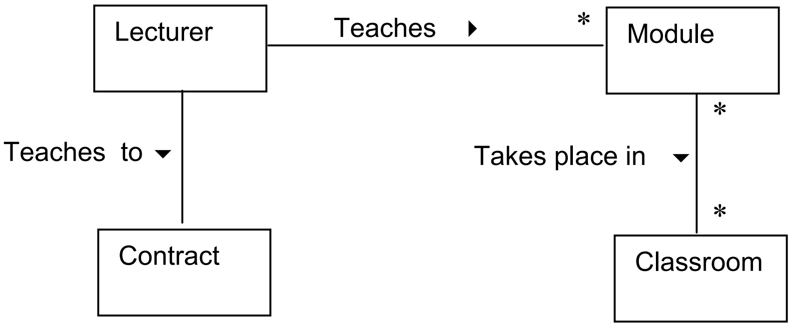


Numbers/ranges are used to indicate known associations

This is accepted UML notation used in OOP design. Associations should be given a text label to describe the relationship between the objects, and a small arrow head can be added to the label to indicate the direction of the relationship.

Example:

Course Timetabling System



There are a number of 'one to many' associations; lecturer and the course module(s). One lecturer teaches many modules, but each module has only one lecturer.

This kind of link would probably be implemented in both directions, so that the timetabling system might allow us to query the lecturer object for its associated modules, but also to query a given module for its associated lecturer object.

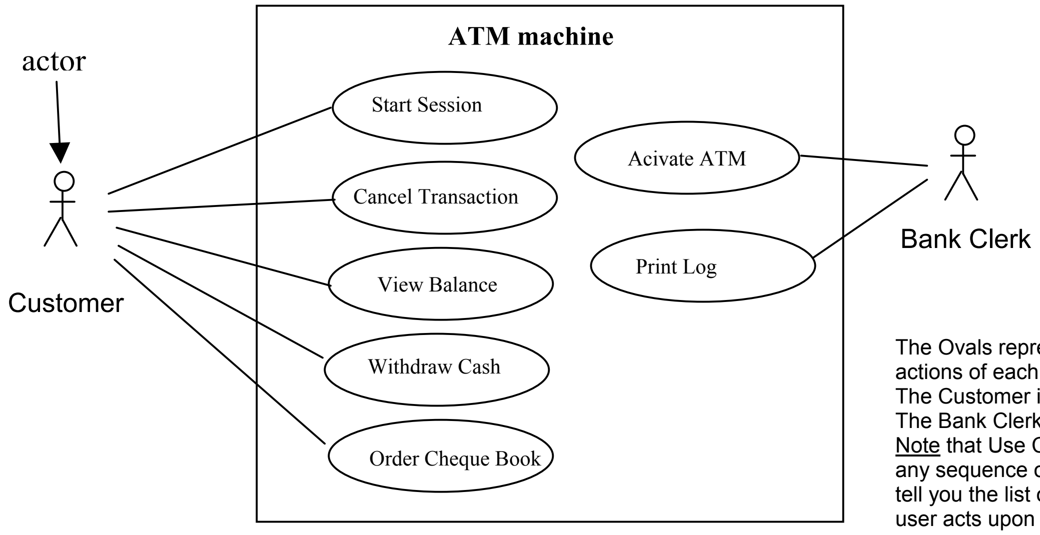
There are also several many to many associations; like an association between modules and rooms (one module may be taught in many classrooms, and each classroom plays host to many different modules).

A one to one association will exist between a lecturer and a teaching contract (so that each lecturer has one contract) and each contract applies to an individual lecturer.

Behaviour

Use Case Diagram

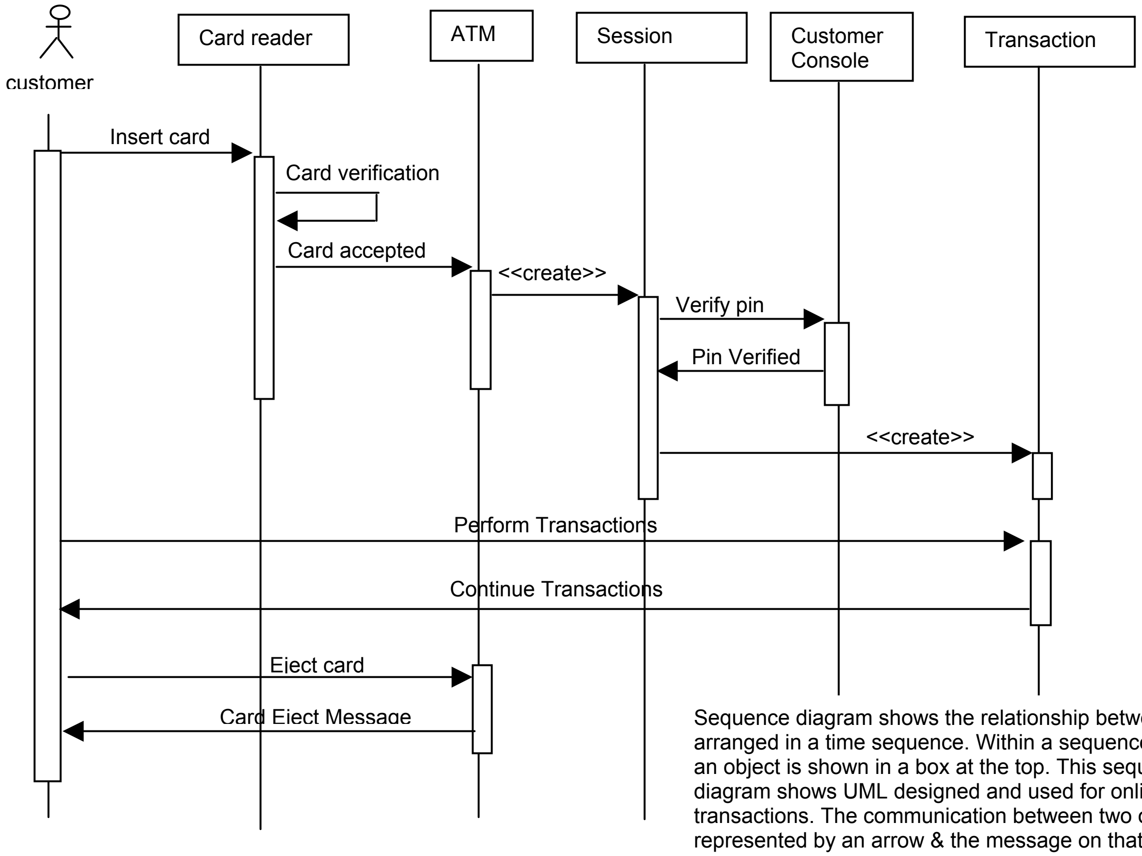
Each ***Use Case*** could be thought of as having a sequence scenario behind it.
A Use Case is acted upon by a ***user*** or ***actor***.
Below is a use case for an ATM machine:



The Ovals represent the different actions of each of the actors.
The Customer interacts with the ATM
The Bank Clerk Activates the ATM
Note that Use Case does not represent any sequence or time events, they just tell you the list of cases and how each user acts upon that case.

Sequence Diagram

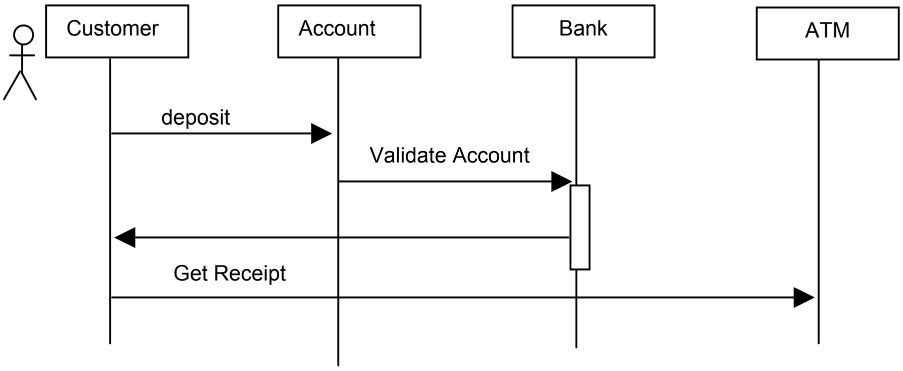
The ***sequence diagram*** describes the flow of messages being passed from object to object. Unlike the class diagram, the sequence diagram represents dynamic message passing between instances of classes rather than just a static structure of classes.
Below is a sample sequence diagram



Sequence diagram shows the relationship between classes arranged in a time sequence. Within a sequence diagram an object is shown in a box at the top. This sequence diagram shows UML designed and used for online money transactions. The communication between two objects represented by an arrow & the message on that arrow, the vertical lines show the life of the objects.

Collaboration Diagram

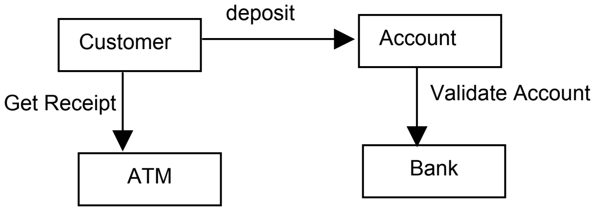
Collaboration is a name given to the interaction among two or more different classes. Consider the sample sequence diagram below:



We can identify three distinct collaborations on this diagram:

- **"Deposit"** – "Customer" communicates to "Account";
- **"ValidateAccount"** – "Account" communicates to "Bank";
- **"GetReceipt"** – "Customer" communicates to "ATM";

Thus, equivalent ***collaboration diagram*** would look like below:



the boxes represent the ***roles*** and the arrows are ***actions*** to be performed by objects with other roles.

- "Customer" collaborate with "MyAccount" by means of "Deposit" communication.
- "MyAccount" collaborate with "Bank" by means of "ValidateAccount" communication.
- "Customer" collaborate with "ATM" by means of "GetReceipt" communication;

Exercise 2.3 Simple Scenario: The Café

Customers come into the café and take a seat at an empty table (each table is numbered). The Customer makes a selection from the Menu and places his order with the Waitress when she visits the table.

The Waitress completes an order slip with the Menu Items and notes the table number. The Order Slip is then passed over to the Cook who prepares the order and plates it up.

The Waitress collects the plate(s) and Order Slip from the Cook. She places the Order Slip next to the till and delivers the plated food to the customer at the numbered table.

When the customer has finished his meal he calls for the bill. The Waitress totals up the cost of the meal and creates a receipt. This receipt is given to the customer and the customer settles the bill and leaves.

1) Draw a use case diagram
2) Draw a sequence diagram
3) Draw a collaboration diagram