

1. An Introduction to Object Orientated Programming

Introduction

This chapter will discuss different programming paradigms and the advantages of the Object Oriented approach to software development and modelling. The concepts on which object orientation depend (abstraction, encapsulation, inheritance and polymorphism) will be explained.

Objectives

By the end of this chapter you will be able to....

- Explain what Object Oriented Programming is,
- Describe the benefits of the Object Oriented programming approach and
- Understand and the basic concepts of abstraction, encapsulation, generalisation and polymorphism on which object oriented programming relies.

All of these issues will be explored in much more detail in later chapters of this book.

This chapter consists of six sections :-

- 1) A Brief History of Computing
- 2) Different Programming Paradigms
- 3) Why use the Object Oriented Paradigm
- 4) Object Oriented Principles
- 5) What Exactly is Object Oriented Programming?
- 6) The Benefits of the Object Oriented Programming Approach.

1.1 A Brief History of Computing

Computing is a constantly changing our world and our environment. In the 1960s large machines called mainframes were created to manage large volumes of data (numbers) efficiently. Bank account and payroll programs changed the way organisations worked and made parts of these organisations much more efficient. In the 1980s personal computers became common and changed the way many individuals worked. People started to own their own computers and many used word processors and spreadsheets applications (to write letters and to manage home accounts). In the 1990s email became common and the world wide web was born. These technologies revolutionised communications allowing individuals to publish information that could easily be accessed on a global scale. The ramifications of these new technologies are still not fully understood as society is adapting to opportunities of internet commerce, new social networking technologies (twitter, facebook, myspace, online gaming etc) and the challenges of internet related crime.

Just as new computing technologies are changing our world so to are new techniques and ideas changing the way we develop computer systems. In the 1950s the use machine code (unsophisticated, complex and machine specific) languages were common.

In the 1960s high level languages, which made programming simpler, became common. However these led to the development of large complex programs that were difficult to manage and maintain.

In the 1970s the structured programming paradigm became the accepted standard for large complex computer programs. The structured programming paradigm proposed methods to logically structure the programs developed into separate smaller, more manageable components. Furthermore methods for analysing data were proposed that allowed large databases to be created that were efficient, preventing needless duplication of data and protected us against the risks associated with data becoming out of sync. However significant problems still persisted in a) understanding the systems we need to create and b) changing existing software as users requirements changed.

In the 1980s ‘modular’ languages, such as Modula-2 and ADA were developed that became the precursor to modern Object Oriented languages.

In the 1990s the Object Oriented paradigm and component-based software development ideas were developed and Object Oriented languages became the norm from 2000 onwards.

The object oriented paradigm is based on many of the ideas developed over the previous 30 years of abstraction, encapsulation, generalisation and polymorphism and led to the development of software components where the operation of the software and the data it operates on are modelled together. Proponents of the Object Oriented software development paradigm argue that this leads to the development of software components that can be re-used in different applications thus saving significant development time and cost savings but more importantly allow better software models to be produced that make systems more maintainable and easier to understand.

It should perhaps be noted that software development ideas are still evolving and new agile methods of working are being proposed and tested. Where these will lead us in 2020 and beyond remains to be seen.

1.2 Different Programming Paradigms

The structured programming paradigm proposed that programs could be developed in sensible blocks that make the program more understandable and easier to maintain.

Activity 1

Assume you undertake the following activities on a daily basis. Arrange this list into a sensible order then split this list into three blocks of related activities and give each block a heading to summarise the activities carried out in that block.

Get out of bed
 Eat breakfast
 Park the car
 Get dressed
 Get the car out of the garage
 Drive to work
 Find out what your boss wants you to do today
 Feedback to the boss on today's results.
 Do what the boss wants you to do

Feedback 1

You should have been able to organise these into groups of related activities and give each group a title that summarises those activities.

Get up :-

Get out of bed
 Get dressed
 Eat breakfast

Go to Work :-

Get the car out of the garage
 Drive to work
 Park the car

Do your job :-

Find out what your boss wants you to do today
 Do what the boss wants you to do
 Feedback to the boss on today's results.

By structuring our list of instructions and considering the overall structure of the day (Get up, go to work, do your job) we can change and improve one section of the instructions without changing the other parts. For example we could improve the instructions for going to work....

Listen to the local traffic and weather report
 Decide whether to go by bus or by car
 If going by car, get the car and drive to work.
 Else walk to the bus station and catch the bus

without worrying about any potential impact this may have on 'getting up' or 'doing your job'. In the same way structuring computer programs can make each part more understandable and make large programs easier to maintain.

The Object Oriented paradigms suggest we should model instructions in a computer program with the data they manipulate and store these as components together. One advantage of doing this is we get reusable software components.

Activity 2

Imagine and a personal address book with some data stored about your friends

Name,
Address,
Telephone Number.

List three things that you may do to this address book.

Next identify someone else who may use an identical address book for some purpose other than storing a list of friends.

Feedback 2

With an address book we would want to be able to perform the following actions :- find out details of a friend i.e. their telephone number, add an address to the address book and, of course, delete an address.

We can create a simple software component to store the data in the address book (i.e. list of names etc) and the operations we can perform (i.e add address, find telephone number etc).

By creating a simple software component to store and manage addresses of friends we can reuse this in another software system i.e. it could be used by a business manager to store and find details of customers. It could also become part of a library system to be used by a librarian to store and retrieve details of the users of the library.

Thus in object oriented programming we can create re-usable software components (in this case an address book).

The Object Oriented paradigm builds upon and extends the ideas behind the structured programming paradigm of the 1970s.

1.3 Why use the Object Orientation Paradigm?

While we can focus our attention on the actual program code we are writing, whatever development methodology is adopted, it is not the creation of the code that is generally the source of most problems. Most problems arise from :-

- poor analysis and design: the computer system we create doesn't do the right thing.
- poor maintainability: the system is hard to understand and revise when, as is inevitable, requests for change arise.

Statistics show 70% of the cost of software is not incurred during its initial development phase but is incurred during subsequent years as the software is amended to meet the ever changing needs of the organisation for which it was developed. For this reason it is essential that software engineers to everything possible to ensure that software is easy to maintain during the years after its initial creation.

The Object Oriented programming paradigm aims to help overcome these problems by helping with the analysis and design tasks during the initial software development phase (see chapter 6 for more details on this) and by ensuring software is robust and maintainable (see chapters 9 -11 for information on the support Object Orientation and Java provides).

1.4 Object Oriented Principles

Abstraction and encapsulation are fundamental principles that underlie the Object Oriented approach to software development. Abstraction allows us to consider complex ideas while ignoring irrelevant detail that would confuse us. Encapsulation allows us to focus on what something does without considering the complexities of how it works.

Activity 3

Consider your home and imagine you were going to swap your home for a week with a new friend.

Write down three essential things you would tell them about your home and that you would want to know about their home.

Now list three irrelevant details that you would not tell your friend.

Feedback 3

You presumably would tell them the address, give them a basic list of rooms and facilities (e.g. number of bedrooms) and tell them how to get in (i.e which key would operate the front door and how to switch off the burglar alarm (if you have one).

You would not tell them irrelevant details (such as the colour of the walls, seats etc) as this would overload them with useless information.

Abstraction allows us to consider the important high level details of your home, e.g. the address, without becoming bogged down in detail.

Activity 4

Consider your home and write down one item, such as a television, that you use on a daily basis (and briefly describe how you operate this item).

Now consider how difficult it would be to describe the internal components of this item and give full technical details of how it works.

Feedback 4

Describing how to operate a television is much easier than describing its internal components and explaining in detail exactly how it works. Most people do not even know all the components of the appliances they use or how they work – but this does not stop them from using appliances every day.

You may not know the technical details such as how the light switches are wired together and how they work internally but you can still switch the lights on and off in your home (and in any new building you enter).

Encapsulation allows us to consider what a light switch does, and how we operate it, without needing to worry about the technical detail of how it actually works.

Two other fundamental principles of Object Orientation are Generalization/specialization (which allows us to make use of inheritance) and polymorphism.

Generalisation allows us to consider general categories of objects which have common properties and then define specialised sub classes that inherit the properties of the general categories.

Activity 5

Consider the people who work in a hospital - list three common occupations of people you would expect to be employed there.

Now for each of these common occupations list two or three specific categories of staff.

Feedback 5

Depending upon your knowledge of the medical profession you may have listed three very general occupations (e.g. doctor, nurse, cleaner) or you may have listed more specific occupations such as radiologist, surgeon etc.

Whatever your initial list you probably would have been able to specify more specialised categories of these occupations e.g.

Doctor :-

Trainee doctor,
Junior doctor,
Surgeon,
Radiologist,
etc

Nurse :-

Triage nurse,
Midwife,
Ward sister

Cleaner :-

General cleaner
Cleaning supervisor

Now we have specified some general categories and some more specialised categories of staff we can consider the general things that are true for all doctors, all nurses etc.

Activity 6

Make one statement about doctors that you would consider to be true for all doctors and make one statement about surgeons that would **not** be true for all doctors.

Feedback 6

You could make a statement that all doctors have a knowledge of drugs, can diagnose medical conditions and can prescribe appropriate medication.

For surgeons you could say that they know how to use scalpels and other specialised pieces of equipment and they can perform operations.

According to our list above all surgeons are doctors and therefore still have a knowledge of medical conditions and can prescribe appropriate medication. However not all doctors are surgeons and therefore not all doctors can perform operations.

What ever we specify as true for doctors is also true for trainee doctors, junior doctors etc – these specialised categories (or classes) can inherit the attributes and behaviours associated with the more general class of ‘doctor’.

Generalisation / specialisation allows us to define general characteristics and operations of an object and allows us to create more specialised versions of this object. The specialised versions of this object will automatically inherit all of the characteristics of the more generalised object.

The final principle underlying Object Orientation is Polymorphism which is the ability to interact with a object as its generalized category regardless of its more specialised category.

Activity 7

Make one statement about how a hospital manager may interact with all doctors employed at their hospital irrespective of what type of doctor they are.

Feedback 7

You may have considered that a hospital manager could pay all doctors (presumably this will be done automatically at the end of every month) and could discipline any doctor guilty of misconduct – of course this would be true for other staff as well. More specifically a manager could check that a doctors medical registration is still current. This would be something that management would need to do for all doctors irrespective of what their specialism is.

Furthermore if the hospital employed new specialist doctor (e.g. a Neurologist), without knowing anything specific about this specialism, hospital management would still know that a) these staff needed to be paid and b) their medical registration must be checked. i.e. they are still doctors and need to be treated as such.

Using the same idea polymorphism allows computer systems to be extended, with new specialised objects being created, while allowing current parts of the system to interact with new object without concern for the specific properties of the new objects.

1.5 What Exactly is Object Oriented Programming?

Activity 8

Think of an object you possess. Describe its current state and list two or three things you can do with that object.

Feedback 8

You probably thought about an entirely physical object such as a watch, a pen, or a car.

Objects have a current status. A watch has a time (represented internally by wheels and cogs or in an electronic component). A pen has a certain amount of ink in it and has its lid on or off. A car has a current speed and has a certain amount of fuel inside it.

Specific behaviour can also be associated with each object (things that you can do with it) :- a watch can be checked to find out its time, its time can also be set. A pen can be used to write with and a car can be driven.

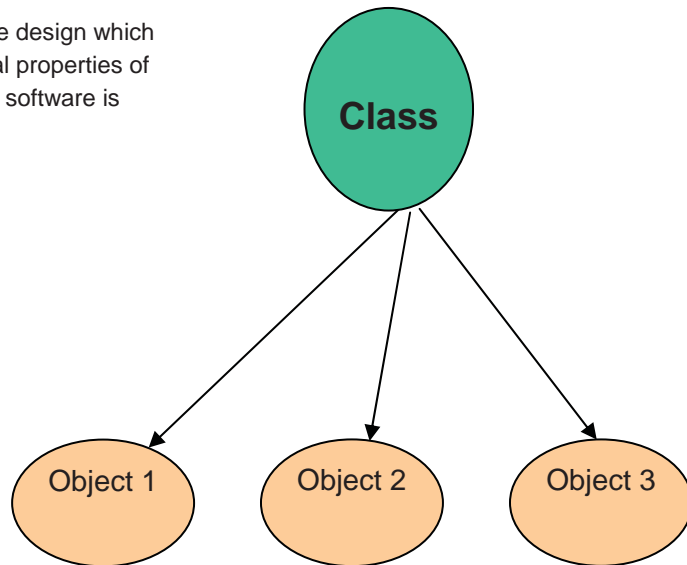
You can also think of other non physical things as objects :- such as a bank account. A bank account is not something that can be physically touched but intellectually we can consider a bank account to be an object. It also has a current status (the amount of money in it) and it also has behaviour associated with it (most obviously deposit money and withdraw money).

Object oriented programming is a method of programming that involves the creation of intellectual objects that model a business problem we are trying to solve (e.g. a bank account, a bank customer and a bank manager – could all be objects in a computerised banking system). With each object we model the data associated with it (i.e. its status at any particular point in time) and the behaviour associated with it (what our computer program should allow that object to do).

In creating an object oriented program we define the properties of a class of objects (e.g. all bank accounts) and then create individual objects from this class (e.g. your bank account).

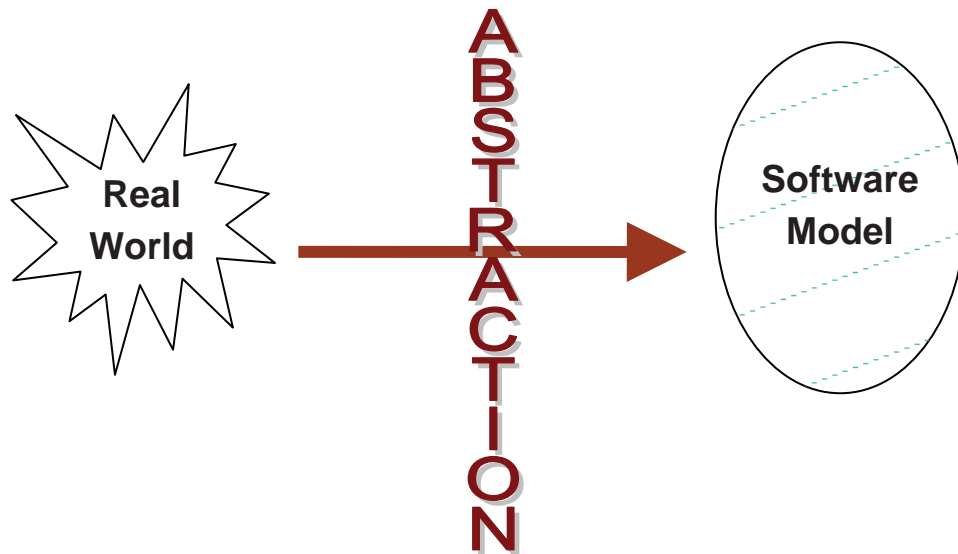
A 'class' is a software design which describes the general properties of something which the software is modelling.

Individual 'objects' are created from the class design for each actual thing



However deciding just what classes we should create in our system is not a trivial task as the real world is complex and messy (see chapter 6 for more advice on how to go about this). In essence we need to create an abstract model of the real world that focuses on the essential aspects of a problem and ignores irrelevant complexities. For example in the real world bank account holders sometimes need to borrow money and occasionally their money may get stolen by a pick pocket. If we were to create a bank account system should we allow customers to borrow money? Should we acknowledge that their cash may get stolen and build in some method of them getting an immediate loan – or is this an irrelevant detail that would just add complexity to the system and provide no real benefit to the bank?

Using object oriented analysis and design techniques our job would be to look at the real world and come up with a simplified abstract model that we could turn into a computer system. How good our final system is will depend upon how good our software model is.

**Activity 9**

Consider a computer system that will allow items to be reserved from a library. Imagine one such item that you may like to reserve and list two or three things that a librarian may want to know about this item.

Feedback 9

You may have thought of a book you wish to reserve in which case the librarian may need to know the title of the book and its author.

Thus for every object we create in a system we need to define the attributes of that object i.e. the things we need to know about it.

Activity 10

Note that we can consider a reservation to be is an intellectual object (where the actual item is a physical object). Considering this intellectual object (item reservation) list two or three actions the librarian may need to perform on this object.

Feedback 10

The librarian may need to cancel this reservation (if you change your mind) they may also need to tell you if the item has arrived in stock for you to collect.

Thus for each object we need to define the operations that will be performed on that object (as well as its attributes).

Activity 11

Considering the most general category of object that can be borrowed from a library, a 'loan item', list two or three more specific subcategory of object a library can lend out.

Feedback 11

Having defined the most general category of object (we call this a class) – something that can be borrowed – we may want to define more specialised sub-classes (e.g. books, magazines, audio/visual material). These will share the attributes defined for the general class but will have specific differences (for example there could be a charge for borrowing audio/visual items).

1.6 The Benefits of the Object Oriented Programming Approach

Whether or not you develop programs in an object oriented way, before you write the software you must first develop a model of what that software must be able to do and how it should work. Object oriented modelling is based on the ideas of abstraction, encapsulation, inheritance and polymorphism.

The general proponents of the object oriented approach claims that this model provides:

- better abstractions (modelling information and behaviour together)
- better maintainability (more comprehensible, less fragile)
- better reusability (classes as encapsulated components)

We will return to look at these claims later in Chapter 11 as we see a case study showing in detail how object oriented analysis works and how the resultant models can be implemented in an object oriented programming language (i.e. Java).

1.7 Summary

Object oriented programming involves the creation of classes by modelling the real world. This allows more specialised classes to be created that inherit the behaviour of the generalised classes.

Polymorphic behaviour means that systems can be changed, as business needs change, by adding new specialised classes and these classes can be accessed by the rest of the system without any regard to their specialised behaviour and without changing other parts of the current system.