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 the basic concepts of abstraction, encapsulation, generalisation and polymorphism on which object oriented programming relies.
- Understand the reasons behind the development of the .NET framework and the role of the Common Language Runtime (CLR) engine.

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

This chapter consists of nine 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.
7) Software Implementation
8) An Introduction to the .NET Framework
9) Summary

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 too 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

"I studied English for 16 years but... ...I finally learned to speak it in just six lessons"
Jane, Chinese architect
ENGLISH OUT THERE
Click to hear me talking before and after my unique course download

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 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, things we can do with the address book (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 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 do 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 C# provides for creating systems that are robust and maintainable).

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.

Whatever 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 allow us to define general characteristics and operations of an object and allow 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.

Excellent Economics and Business programmes at:
university of groningen
AACSB ACCREDITED
"The perfect start of a successful, international career."
CLICK HERE to discover why both socially and academically the University of Groningen is one of the best places for a student to be
www.rug.nl/feb/education

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 doctor's 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 a 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.

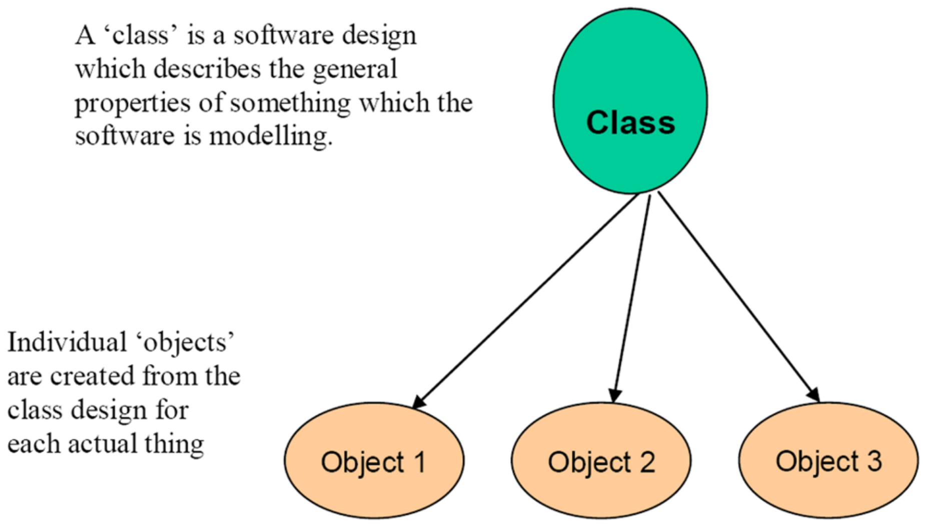
Objects have a current status. A watch has a time (represented internally by wheels and cogs or in an electronic component). A pen has a certain amount of ink in it and has its lid on or off. A car has a current speed and has a certain amount of fuel inside it.

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, started, driven and stopped.

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).

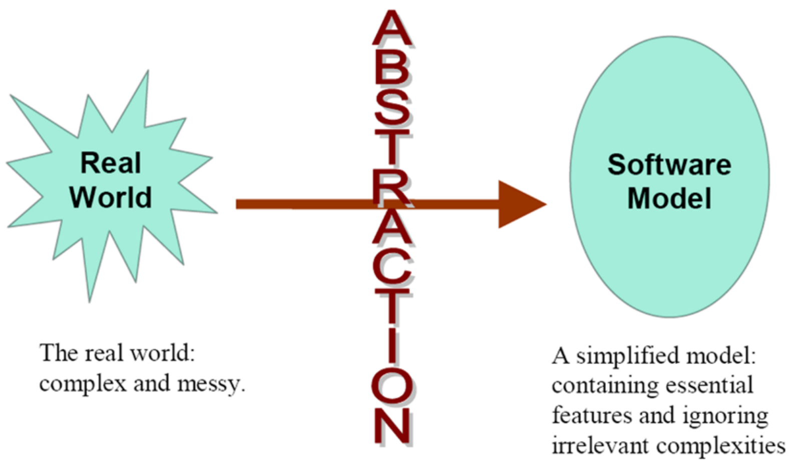
Object oriented programming it a method of programming that involves the creation of intellectuals objects that model a business problem we are trying to solve (e.g. a bank account, a bank customer and a bank manager – could all be objects in a computerised banking system). With each object we model the data associated with it (i.e. it 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 a 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.
Class
Individual 'objects' are created from the class design for each actual thing
Object 1
Object 2
Object 3

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 a 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.


Real World
ABSTRACTION
Software Model
The real world: complex and messy.
A simplified model: containing essential features and ignoring irrelevant complexities

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.

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 as 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.

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 software)
- better reusability (classes as encapsulated components that can be used in other systems)

We will look at these claims throughout this book and in Chapter 11 we will 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. C#).

1.7 Software Implementation

Before a computer can complete useful tasks for us (e.g. check the spelling in our documents) software needs to be written and implemented on the machine it will run on. Software implementation involves the writing of program source code and preparation for execution of this on a particular machine. Of course before the software is written it needs to be designed and at some point it needs to be tested. There are many iterative lifecycles to support the process of design, implementation and testing that involve multiple implementation phases. Of particular concern here are the three long established approaches to getting source code to execute on a particular machine…

- compilation into machine-language object code
- direct execution of source code by 'interpreter' program
- compilation into intermediate object code which is then interpreted by run-time system

Implementing C# programs involves compiling the source code (C#) into machine-language object code. This approach has some advantages and disadvantages and it is worth comparing these three options in order to appreciate the implications for the C# developer.

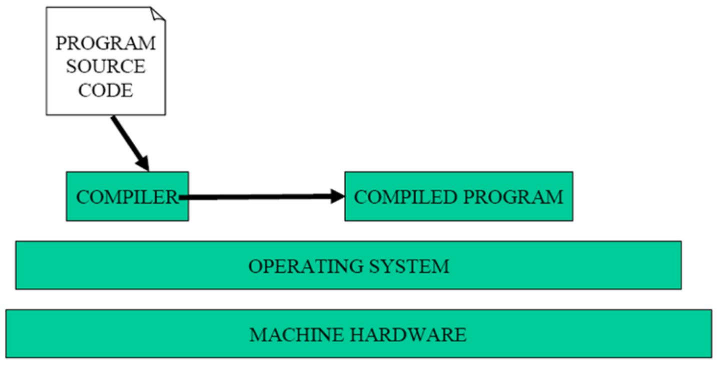
Compilation

The compiler translates the source code into machine code for the relevant hardware / operating system combination.

Strictly speaking there are two stages: compilation of program units (usually files), followed by 'linking' when the complete executable program is put together including the separate program units and relevant library code etc.

The compiled program then runs as a 'native' application for that platform.

This is the oldest model, used by early languages like Fortran and Cobol, and many modern ones like C#. It allows fast execution speeds but requires re-compilation of the program each time the code is changed or each time we want to run this code on a machine with a different operating system.



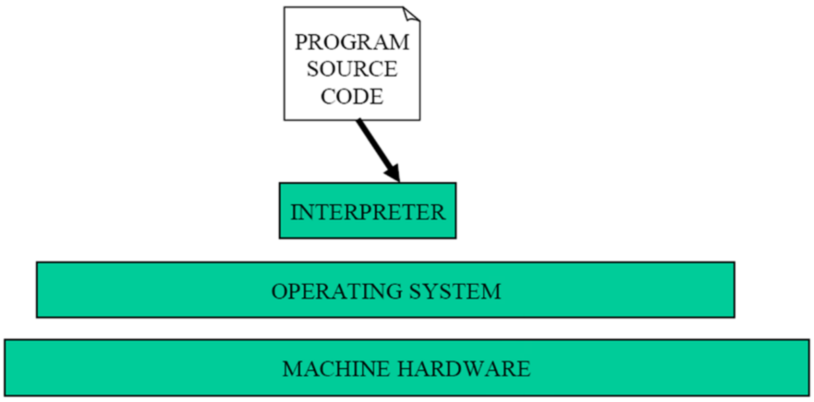
Interpretation

Here the source code is not translated into machine code. Instead an interpreter reads the source code and performs the actions it specifies.

We can say that the interpreter is like a 'virtual machine' whose machine language is the source code language.

No re-compilation is required after changing the code, but the interpretation process inflicts a significant impact on execution speed.

Scripting languages tend to work in this way.



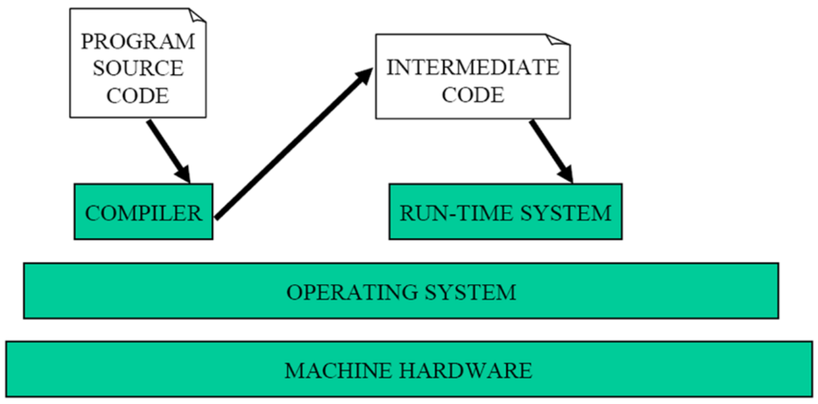
Interpreted programs can be slow but can work on any machine that has an appropriate interpreter. They do not need to be complied for different machines.

Intermediate Code

This model is a hybrid between the previous two.

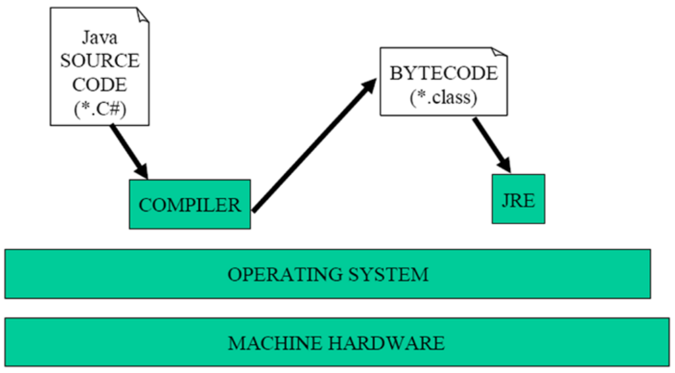
Compilation takes place to convert the source code into a more efficient intermediate representation which can be executed by a 'run-time system' (again a sort of 'virtual machine') more quickly that direct interpretation of the source code. However, the use of an intermediate code which is then executed by run-time system software allows the compilation process to be independent of the operating system / hardware platform, i.e. the same intermediate code should run on different platforms so long as an appropriate run-time system is available for each platform.

This approach is long-established (e.g. in Pascal from the early 1970s) and is how Java works. Java is a modern Object Oriented Language which is an alternative to C# and yet shares many similar features from the programmers point of view.



Running Java Programs

To run Java programs we must first generate intermediate code (called bytecode) using a compiler available as part of the Java Development Kit (JDK). Thus a Java compiler does not create .exe files i.e. code that could run directly on a specific machine but instead generates .class files.



A version of the Java Runtime Environment (JRE), which incorporates a Java Virtual machine (VM), is required to execute the bytecode and the Java library packages. Thus a JRE must be present on any machine which is to run Java programs.

The Java bytecode is standard and platform independent and as JRE's have been created for most computing devices (including PC's, laptops, mobile devices, mobile phones, internet devices etc) this makes Java programs highly portable and by compiling the code to an intermediate language Java strives to attain the fast implementation speeds obtained by fully compiled systems. i.e. Once complied Java code can run on any machine with a JRE irrespective of its underlying operating system without needing to be recompiled.

Running C# Programs

As we will see in section 1.8, C# programs, like Java programs, are also converted into an intermediate language but a C# compiler then goes further and links these with the necessary dll files to generate programs that can be directly executed on the target machine. C# programs are therefore fully complied programs i.e. .exe files are generated. While these are fast and efficient the source code must be recompiled each time we wish to run the program on a machine with a different operating system.

C# it is part of the .NET framework that aims to deliver additional benefits for the programmer.

Some programmers mistakenly believe that C# was written to create programmes only for the Windows operating system this is not true as we will see when we consider the .NET framework.

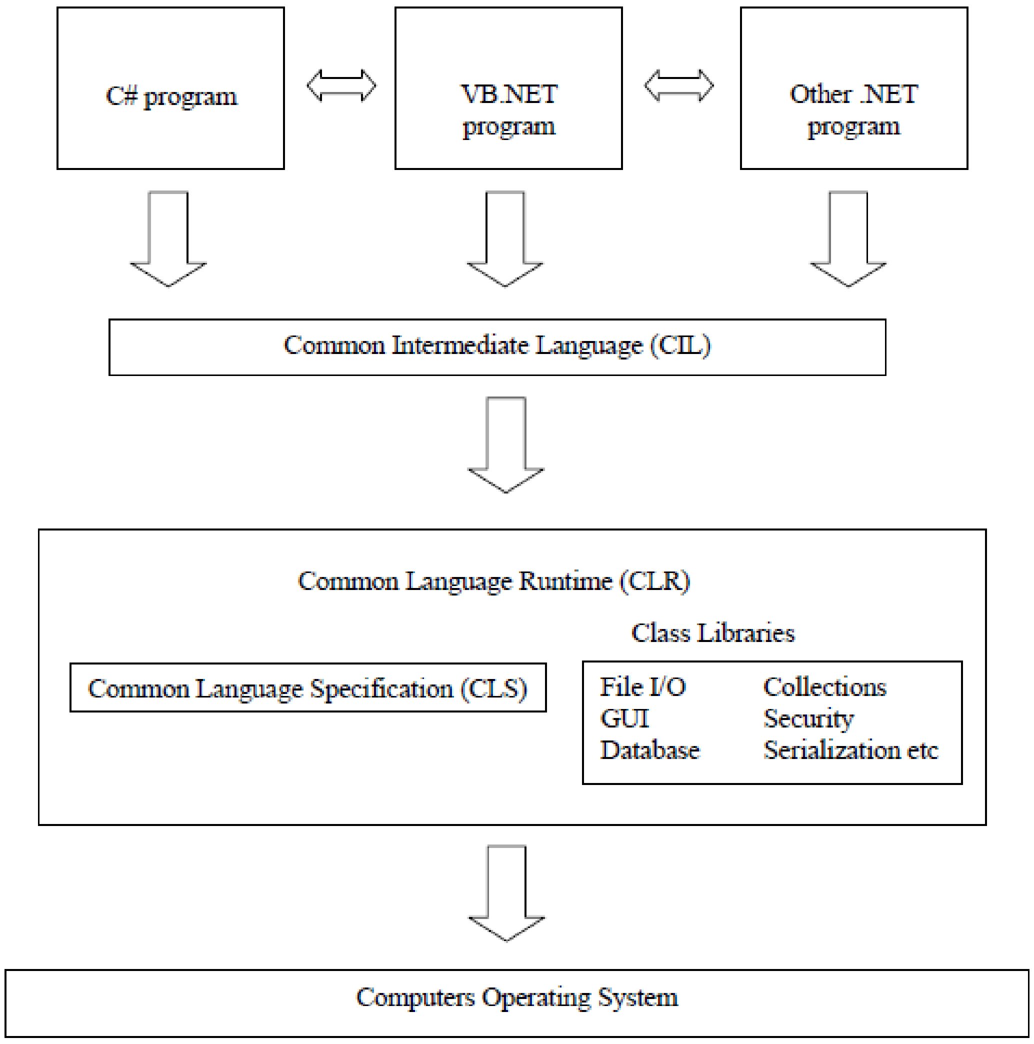
1.8 An Introduction to the .NET Framework

The .NET framework was designed to make life simpler for programmers by providing a common platform that can be used when writing programs in several different programming languages (C# and Visual Basic among others).

The .NET platform provides common features that can be used by programmers of all .NET languages. These include :-

a) a Common Language Runtime (CLR) engine that allows all .NET programs to run and use common features based on a Common Language Specification (CLS). The CLR also shields the programmer from having to deal with issues that are specific to individual operating systems.
b) a comprehensive class library providing a wealth of in built functionality (GUI, File I/I, Database, Threading, Serialization, Security, Web applications, Networking etc)
c) a Common Intermediate Language (CIL). All .NET source code is compiled into a common intermediate language this provides the ability to integrate code written in any .NET language making use of common exception handling facilities.

This relationship can be shown diagrammatically as below :-


C# program
VB.NET program
Other .NET program
Common Intermediate Language (CIL)
Common Language Runtime (CLR)
Class Libraries
Common Language Specification (CLS)
File I/O
GUI
Database
Collections
Security
Serialization etc
Computers Operating System

C# programs are thus partly compiled into a common intermediate language. This is then linked with a CLR engine to give a .exe file.

Taken all together the .NET framework does not mean that all of the .NET languages are identical but they do have access to an identical library of methods that can be used by all .NET Languages. Thus learning to programme one .NET language makes it easier to learn and program another .NET language.

.NET programs can only run where a CLR engine has been created for the underlying operating system. Windows Vista and Windows 7 comes with a CLR engine and can therefore run .NET programs but older versions of windows may need a CLR engine installed before .NET programs can be run.

To enable .NET programs to be platform independent Microsoft published an open source specification for a CLR engine (called a Virtual Execution System) this included the definition of the C# language and a Common Language Infrastructure (CLI).

By using these definitions CLR engines have been created for other operating systems (e.g. Mac OS and Linux) and by using these .NET programs can run on different platforms… not just Microsoft Windows. However before programs can be run on these different platforms they do need to be recompiled for each platform thus .NET programs, including C# programs, are not as portable as Java programs.

While CLR engines are perhaps not as widely available as JREs they do exist for other platforms. One example of an open source CLR is Mono (www.mono-project.com). Mono is an open source, cross-platform, implementation of C# and the CLR that is compatible with Microsoft.NET. One part of the Mono project is MonoTouch. MonoTouch allows you to create C# and .NET apps for iPhone and iPod Touch, while taking advantage of iPhone APIs.

As well as compiling our C# programs it is possible to create a software installation routine that will download via the web a .NET runtime environment and automatically install this along with our software (assuming a .NET runtime environment is not already installed).

We will be writing and running code written in C# throughout this book and in doing so we will be making use of a compiler, the Common Language Runtime (CLR) engine and some of the more common class libraries. However the prime aim of this book is to teach generic Object Oriented programming and modelling principles that are common across a range of OO languages – not just .NET languages.

While we will illustrate OO principles in this book with C# code we will not concern ourselves further with the intricacies of how the CLR engine works, details regarding how .NET programs are compiled nor the detailed operation of the Common Intermediate Language (CIL).

However in order that the examples illustrated in this book can be demonstrated as practical worked examples we will introduce two modern Interactive Development Environments (IDEs), and some other tools specifically designed for the creation of .NET programs (see Chapter 8).

1.9 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.

We will return to each of the concepts introduced here throughout the book and hopefully, by the end, you will have a good understanding of these concepts and understand how to apply them using C#.

Maastricht University
Leading in Learning!
Join the best at the Maastricht University School of Business and Economics!
Top master's programmes
• 33rd place Financial Times worldwide ranking: MSc International Business
• 1st place: MSc International Business
• 1st place: MSc Financial Economics
• 2nd place: MSc Management of Learning
• 2nd place: MSc Economics
• 2nd place: MSc Econometrics and Operations Research
• 2nd place: MSc Global Supply Chain Management and Change
Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012
Maastricht University is the best specialist university in the Netherlands (Elsevier)
Visit us and find out why we are the best!
Master's Open Day: 22 February 2014
www.mastersopenday.nl