Object-Oriented Programming Concepts

by Richard Carr, published at http://www.blackwasp.co.uk/ObjectOrientedConcepts.aspx

This is the first in a series of articles describing the use of object-oriented programming techniques as implemented by the C# programming language. This series follows the C# Fundamentals tutorial that beginners are advised to read first.

Tutorial Prerequisites

The .NET Framework includes all of the software required to compile a program that can be developed using any standard text editor. However, Microsoft provides many development environments that are suitable and make programming and debugging much easier.

Microsoft Visual Studio is an ideal development environment for this tutorial if you have it available. If not, I strongly suggest downloading Microsoft's excellent, *and free-of-charge*, development environment, Microsoft C# Express Edition . However, if you have another preferred development environment for C# programming, you should easily be able to adapt the examples accordingly.

What is Object-Oriented Programming?

As computers increase in processing power, the software they execute becomes more complex. This increased complexity comes at a cost of large programs with huge codebases that can quickly become difficult to understand, maintain and keep bug-free.

Object-oriented programming (OOP) tries to alleviate this problem by creating networks of objects, each like a small software 'machine'. These objects are naturally smaller entities, simplifying the development task of each unit. However, when the objects co-operate in a system, they become the building blocks of much more complex solution.

Consider the motor car. If a car were designed as a single machine, it would be seen as hugely complex with lots of opportunities for failure. However, when broken down into its constituent parts, such as wheels, pedals, doors, etc. the individual design items become simpler. Each part (or object) is created independently, tested carefully and then assembled into the final product. The creation of the parts can be simplified further when they are broken down into even simpler items. For example, when each door is considered as being composed of an outer panel, handle, inner panel and window.

The car example is analogous to the object-oriented software. Rather than writing a huge program to create, for example, a project management system, the solution is broken into real-world parts such as *project*, *task*, *estimate*, *actual*, *deliverable*, etc. Each of these can then be developed and tested independently before being combined.

Key Concepts

This tutorial describes the use of object-oriented techniques with the C# programming language. Before we start to examine any sample code, it is important to review the ideas that will be discussed. Some of the key concepts are described in the following sections.

Classes and Objects

The basic building blocks of object-oriented programming are the *class* and the *object*. A class defines the available characteristics and behaviour of a set of similar objects and is defined by a programmer in code. A class is an abstract definition that is made concrete at run-time when objects based upon the class are instantiated and take on the class's behaviour.

As an analogy, let's consider the concept of a 'vehicle' class. The class developed by a programmer would include methods such as *Steer(), Accelerate()* and *Brake()*. The class would also include properties such as *Colour*, *NumberOfDoors*, *TopSpeed* and *NumberOfWheels*. The class is an abstract design that becomes real when objects such as Car, RacingCar, Tank and Tricycle are created, each with its own version of the class's methods and properties.

Car	Racing Car	Tank	Tricycle
			
Colour: *Red* NumberOfDoors: *2* NumberOfWheels: *4* TopSpeed: *60mph*	Colour: *Yellow* NumberOfDoors: *0* NumberOfWheels: *4* TopSpeed: *200mph*	Colour: *Green* NumberOfDoors: *1* NumberOfWheels: *12* TopSpeed: *40mph*	Colour: *Orange* NumberOfDoors: *0* NumberOfWheels: *3* TopSpeed: *50mph*

Encapsulation

Encapsulation,also known as *data hiding*, is an important object-oriented programming concept. It is the act of concealing the functionality of a class so that the internal operations are hidden, and irrelevant, to the programmer. With correct encapsulation, the developer does not need to understand how the class actually operates in order to communicate with it via its publicly available methods and properties; known as its public *interface*.

Encapsulation is essential to creating maintainable object-oriented programs. When the interaction with an object uses only the publicly available *interface* of methods and properties, the class of the object becomes a correctly isolated unit. This unit can then be replaced independently to fix bugs, to change internal behaviour or to improve functionality or performance.

In the car analogy this is similar to replacing a headlight bulb. As long as we choose the correct bulb size and connection (the public interface), it will work in the car. It does not matter if the manufacturer has changed or the internal workings of the bulb differ from the original. It may even offer an improvement in brightness!

Passing Messages

Message passing, also known as *interfacing*, describes the communication between objects using their public interfaces. There are three main ways to pass messages. These are using *methods*, *properties* and *events*. A property can be defined in a class to allow objects of that type to advertise and allow changing of state information, such as the 'TopSpeed' property. Methods can be provided so that other objects can request a process to be undertaken by an object, such as the Steer() method. Events can be defined that an object can raise in response to an internal action. Other objects can subscribe to these so that they can react to an event occurring. An example for vehicles could be an 'ImpactDetected' event subscribed to by one or more 'AirBag' objects.