**Anonymous classes**

As you might expect, an anonymous class is a class that does not have a name.   Anonymous classes offer the programmer a convenient way of encapsulating read-only properties into a single object without the need to explicitly define a type first. The type name will be generated by the compiler.  The type name is also not available at the source code level and the type of each property included in this anonymous class will be inferred by the compiler.

To create an anonymous class. you simply use the new keyword followed by a pair of braces to define fields and values for the class.  The following is an example:

anAnonymousObject = new { Name = "Tom", Age = 65 };

The class will have two public fields, Name (initialized to the string “Tom”) and Age (initialized to 65). The compiler has inferred the types of these two fields based on the types of data you initialize them with.

Because our anonymous class doesn't have a name,  how can you create an object of that type and assign an instance of the class to it? In the preceding code example, what should the type of the object variable anAnonymousObject be?  As a result of the way anonymous classes work, you don’t know, which is precisely the point of anonymous classes.

This doesn't truly present a problem however, as long as you declare anAnonymousObject as an implicitly typed variable by using the var keyword as shown here:

var anAnonymousObject = new { Name = "Tom", Age = 65 };

Recall that using the var keyword will result in the compiler creating a variable using the same type as the expression that was used to initialize it. In this case, the type of the expression is whatever name the compiler happens to generate for the anonymous class.

Once instantiated, you can access the fields in the object by using dot notation, as shown in this example:

Console.WriteLine("Name: {0} Age: {1}", anAnonymousObject.Name, anAnonymousObject.Age};

Once created, you have the option to create other instances of the same anonymous class but with different values:

var secondAnonymousObject = new { Name = "Hal", Age = 46 };

The C# compiler will look at the names, types, number, and the order of the fields in the object in order to determine whether two instances of an anonymous class have the same type or not.  In our two examples, both objects contain the same number of fields, the same name and the same type, in the same order.  As a result, both variables are instances of the same anonymous class. This means that you can assign anAnonymousObject to the secondAnonymousObject or vice versa:

secondAnonymousObject = anAnonymousObject;

***Note:***  There are quite a few restrictions on the contents of an anonymous class:

* anonymous classes can contain only public fields
* the fields must all be initialized
* fields cannot be static
* you cannot define any methods for them