Herencia de clases

Las clases admiten completamente la *herencia*, una característica fundamental de la programación orientada a objetos. Al crear una clase, puede heredar de cualquier otra interfaz o clase que no esté definida como sealed; y otras clases pueden heredar de su clase e invalidar los métodos virtuales de la clase.

La herencia se consigue mediante una *derivación*, en la que se declara una clase mediante una *clase base*, desde la que hereda los datos y el comportamiento. Una clase base se especifica anexando dos puntos y el nombre de la clase base seguido del nombre de la clase derivada, como en el siguiente ejemplo:

```
public class Manager : Employee
{
    // Employee fields, properties, methods and events are inherited
    // New Manager fields, properties, methods and events go here...
}
```

Cuando una clase declara una clase base, hereda todos los miembros de la clase base excepto los constructores.

A diferencia de C++, una clase de C# solo puede heredar directamente de una clase base. En cambio, dado que una clase base puede heredar de otra clase, una clase podría heredar indirectamente varias clases base. Además, una clase puede implementar directamente más de una interfaz. Para obtener más información, vea el módulo de Interfaces.

Una clase puede declararse abstracta. Una clase abstracta contiene métodos abstractos que tienen una definición de firma, pero no tienen ninguna implementación. No se pueden crear instancias de las clases abstractas. Solo se pueden usar a través de las clases derivadas que implementan los métodos abstractos. Por el contrario, la clase sealed no permite que otras clases se deriven de ella. Las definiciones de clase se pueden dividir entre distintos archivos de código fuente.

Ejemplo

En el ejemplo siguiente se define una clase pública que contiene una propiedad implementada automáticamente, un método y un método especial denominado constructor.

```
using System;

public class Person
{
    // Constructor that takes no arguments:
    public Person()
    {
        Name = "unknown";
    }

    // Constructor that takes one argument:
    public Person(string name)
    {
        Name = name;
    }

    // Auto-implemented readonly property:
    public string Name { get; }

    // Method that overrides the base class (System.Object) implementation.
    public override string ToString()
    {
        return Name;
    }
}
class TestPerson
{
    static void Main()
    {
        // Call the constructor that has no parameters.
        var person1 = new Person();
        Console.WriteLine(person1.Name);

        // Call the constructor that has one parameter.
        var person2 = new Person("Sarah Jones");
        Console.WriteLine(person2.Name);
        // Get the string representation of the person2 instance.
        Console.WriteLine(person2);

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
// Output:
// unknown
// Sarah Jones
// Sarah Jones
```