

Programación Web .NET Core

Módulo 2

Accesibilidad

Accesibilidad - Clases estáticas

Las clases estáticas son aquellas desde las cuales no podemos crear instancias/objetos. Es decir, no puede utilizar la palabra clave **new** para crear una variable de instancia de una clase estática. Ya que no hay ninguna variable de instancia, el acceso a los miembros de una clase estática se realiza mediante el propio **nombre de clase**. Por ejemplo, si tiene una clase estática con el nombre de **Utiles** e incluye un método público llamado **Agregar**, la llamada al método se hace tal como se muestra en el ejemplo siguiente:

```
Utiles.Agregar();
```

Una clase estática se podría utilizar como un contenedor de conjuntos de métodos, que sólo funcionan con parámetros de entrada y no tienen que obtener ni establecer campos internos de instancia. Por ejemplo, en la biblioteca de clases .NET Framework, la clase estática **Math** contiene varios métodos que realizan operaciones matemáticas, sin ningún requisito para almacenar o recuperar datos únicos de una instancia determinada de la clase **Math**. En otras palabras, decimos que los miembros de la clase se aplican al indicar el nombre de la clase y del método, como se muestra en el ejemplo de la siguiente pantalla.

```
doubledub = -3.14;  
Console.WriteLine(Math. Abs(dub));  
Console.WriteLine(Math . Floor(dub));  
Console.WriteLine(Math. Round(Math. Abs(dub)));
```

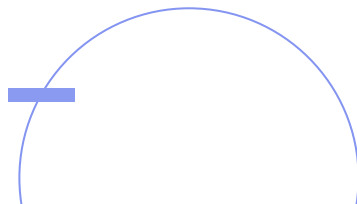


```
C:\Users\mmyszne\source\repos\ConsoleApp4\bin\Debug\ConsoleApp4.exe  
3.14  
-4  
3
```

Como sucede con todos los tipos de clase, **CommonLanguageRuntime (CLR)** de .NET Framework **carga la información de tipo de una clase estática cuando se carga el programa que hace referencia a la clase.** El programa no puede especificar exactamente cuándo se carga la clase. Sin embargo, se garantiza que se ha cargado, que sus campos se han inicializado y que se ha llamado a su constructor estático antes de que se haga referencia a la clase por primera vez en el programa. Solo se llama una vez a un constructor estático y una clase estática permanece en memoria durante el período de duración que corresponde a la aplicación que la utiliza.

Características esenciales de una clase estática:

- Sólo contiene miembros estáticos.
- No se pueden crear instancias de ella (no se puede usar **new**).
- Es de tipo **sealed**, otras clases no pueden heredar de ella.
- No puede contener constructores de instancia.

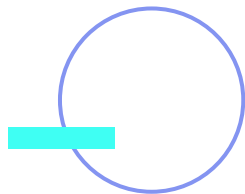


Resumen

En síntesis, crear una **clase estática** es, básicamente igual que crear una clase que **sólo tiene miembros estáticos y un constructor privado**. Un constructor privado evita que se creen instancias de la clase. La ventaja de utilizar una clase estática es que el compilador puede comprobar que no se agregue accidentalmente ningún miembro de instancia. **El compilador garantizará que no se puedan crear instancias de esta clase.**

Las clases estáticas al ser de tipo **sealed** (clases selladas) **no pueden heredarse**. No pueden heredar de ninguna clase, excepto de la clase **Object**.

Las clases estáticas no pueden tener un constructor de instancia; sin embargo, **pueden tener un constructor estático**. Las clases no estáticas también deben definir un constructor estático si contienen miembros estáticos que requieren una inicialización no trivial.



Ejemplo

Este es un ejemplo de una clase estática que contiene dos métodos que convierten la temperatura de grados Celsius a Fahrenheit y viceversa:

```
publicstaticclassConvertidorTemperatura
{
    publicstaticdoubleCelsiusAFahrenheit(stringtemperaturaCelsius)
    {
        // Convierte el argumento a tipo double para hacer los cálculos.
        doublecelsius = Double.Parse(temperaturaCelsius);

        // Convierte de Celsius a Fahrenheit.
        doublefahrenheit = (celsius * 9 / 5) + 32;

        returnfahrenheit;
    }
}
```



```
public static double FahrenheitACelsius(string temperaturaFahrenheit)
{
    // Convierte el argumento a tipo double para hacer los cálculos.
    double fahrenheit = Double.Parse(temperaturaFahrenheit);

    // Convierte Fahrenheit a Celsius.
    double celsius = (fahrenheit - 32) * 5 / 9;

    return celsius;
}
```




```
class PruebaConvertidorTemperatura
{
    static void Main()
    {
        Console.WriteLine("Indique el tipo de conversión a realizar:");
        Console.WriteLine("1. De Celsius a Fahrenheit.");
        Console.WriteLine("2. De Fahrenheit a Celsius.");
        Console.Write(":");

        string seleccion = Console.ReadLine();
        double F, C = 0;
    }
}
```

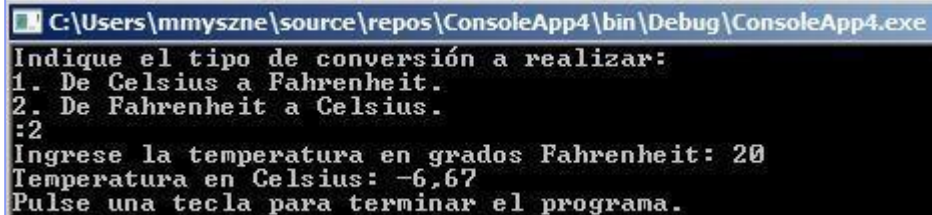
```
switch (seleccion)
{
    case "1":
        Console.Write("Ingrese la temperatura en grados Celsius: ");
        F = ConvertidorTemperatura.CelsiusAFahrenheit(Console.ReadLine());
        Console.WriteLine("Temperatura en Fahrenheit: {0:F2}", F);
        break;

    case "2":
        Console.Write("Ingrese la temperatura en grados Fahrenheit: ");
        C = ConvertidorTemperatura.FahrenheitACelsius(Console.ReadLine());
        Console.WriteLine("Temperatura en Celsius: {0:F2}", C);
        break;

    default:
        Console.WriteLine("Por favor seleccione el convertidor a realizar.");
        break;
}
```

...

```
// Mantiene la consola abierta para ver el resultado.  
    Console.WriteLine("Pulse una tecla para terminar el programa.");  
    Console.ReadKey();  
}  
}
```

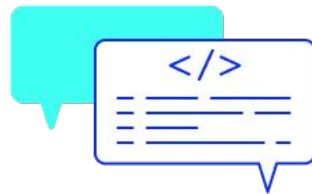


```
C:\Users\mmyszne\source\repos\ConsoleApp4\bin\Debug\ConsoleApp4.exe  
Indique el tipo de conversión a realizar:  
1. De Celsius a Fahrenheit.  
2. De Fahrenheit a Celsius.  
:2  
Ingrese la temperatura en grados Fahrenheit: 20  
Temperatura en Celsius: -6.67  
Pulse una tecla para terminar el programa.
```

Miembros estáticos

Una clase no estática puede contener métodos, campos, propiedades o eventos estáticos. El miembro estático es invocable en una clase, incluso si no se ha creado ninguna instancia de esa clase. **El nombre de clase y no el nombre de instancia, es el que tiene acceso al miembro estático.** Solo existe una copia de un miembro estático, independientemente del número de instancias que se creen de la clase.

Los métodos y propiedades estáticos no pueden tener acceso a los campos y eventos no estáticos de su tipo contenedor y no pueden tener acceso a una variable de instancia de ningún objeto a menos que se pase explícitamente en un parámetro de método.



Es más habitual declarar una clase no estática con algunos miembros estáticos que declarar toda una clase como estática. Dos usos comunes de los campos estáticos son los siguientes: mantener un recuento del número de objetos de los que se han creado instancias y almacenar un valor que se debe compartir entre todas las instancias de una clase.

Los métodos estáticos se pueden sobrecargar, pero no invalidar porque pertenecen a la clase y no a una instancia de la clase.

Aunque un campo no se puede declarar como **staticconst**, un campo **const** es esencialmente estático en su comportamiento. Pertenecce al tipo, no a las instancias del tipo. Por consiguiente, se puede tener acceso a los campos de constante mediante la misma notación llamada **NombreClase.NombreMiembroDeClase** que se utiliza para los campos estáticos. No se requiere ninguna instancia de objeto.



C# no admite el uso de variables locales estáticas (variables declaradas en el ámbito del método).

Los miembros estáticos de la clase se declaran mediante la incorporación de la palabra clave **static** antes del tipo de valor devuelto del miembro, tal como se muestra en el ejemplo siguiente:



```
public class Automovil
{
    public static int CantidadDeRuedas = 4;
    public static int TamanoTanqueCombustible
    {
        get { return 15; }
    }
    public static void Conducir() { }

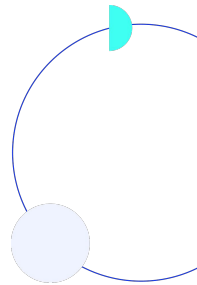
    // Otros campos y propiedades no estáticos...
}
```



Los miembros estáticos se inicializan antes de obtener acceso al miembro estático por primera vez y antes de llamar al constructor estático, si éste existe. Para tener acceso a un miembro estático de la clase, utiliza el nombre de la clase en lugar de un nombre de variable para indicar la ubicación del miembro, tal como se muestra en el ejemplo a continuación:

```
static void Main()  
{  
    Automovil.Conducir();  
    int i = Automovil.CantidadDeRuedas;  
}
```

Si la clase contiene campos estáticos, proporciona un constructor estático que los inicialice cuando se cargue la clase.

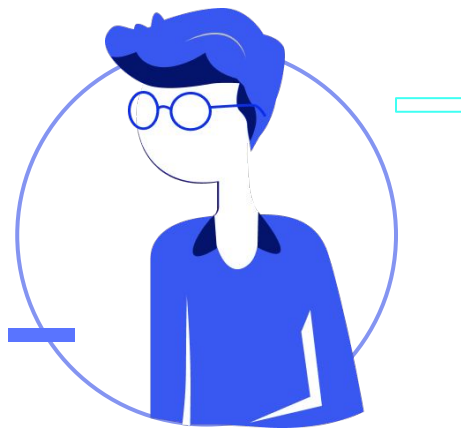


Clases abstractas y clases selladas

La palabra clave **abstract** permite **crear clases y miembros de clase que están incompletos** (como si creara un esqueleto o definición para una clase base), y se deben implementar en una clase derivada.

Es decir, el código de estas clases debe ser codificado en las clases que hereden dicha clase.

La palabra clave **sealed** permite **impedir la herencia** de una clase o de ciertos miembros de clase marcados previamente como virtuales.



Clases y miembros de clase abstractos

Las clases se pueden declarar como abstractas si se incluye la palabra clave **abstract** antes de la definición de clase. Por ejemplo:

```
publicabstractclass A
{
    // Aquí van los miembros de la clase.
}
```

No se pueden crear instancias de una clase abstracta. El propósito de una clase abstracta es proporcionar **una definición común de una clase base que múltiples clases derivadas pueden compartir.** Por ejemplo, una biblioteca de clases puede definir una clase abstracta que se utiliza como parámetro para muchas de sus funciones y solicitar a los programadores que utilizan esa biblioteca que proporcionen su propia implementación de la clase, mediante la creación de una clase derivada de esta clase abstracta.

Las clases abstractas también pueden definir métodos abstractos. Esto se consigue al agregar la palabra clave **abstract** antes del tipo de valor que devuelve el método. Por ejemplo:

```
publicabstractclass A
{
    publicabstractvoidHacerAlgo(int i);
}
```

Los métodos abstractos no tienen ninguna implementación, de modo que la definición de método va seguida por un punto y coma en lugar de un bloque de método normal. Las clases derivadas de la clase abstracta deben implementar todos los métodos abstractos. Cuando una clase abstracta hereda un método virtual de una clase base, la clase abstracta puede reemplazar el método virtual con un método abstracto. Veamos un ejemplo en la siguiente pantalla.



```
publicclass D
{
    publicvirtualvoidHacerAlgo(int i)
    {
        // Implementación Original.
    }
}

publicabstractclassE : D
{
    // Override sin implementación porque es Abstracta.
    publicabstractoverridevoidHacerAlgo(int i);    }

publicclassF : E
{
    publicoverridevoidHacerAlgo(int i)    {
        // nueva implementación
    }
}
```

Si un método virtual se declara como **abstract**, sigue siendo virtual para cualquier clase que herede de la clase abstracta. Una clase que hereda un método abstracto no puede tener acceso a la implementación original del método; en el ejemplo anterior, **HacerAlgo** de la clase **F** no puede llamar a **HacerAlgo** de la clase **D**. De esta forma, una clase abstracta puede obligar a las clases derivadas a proporcionar nuevas implementaciones de método para los métodos virtuales.

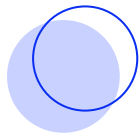


Clases y miembros de clase sellados

Las clases se pueden declarar como selladas si se incluye la palabra clave **sealed** antes de la definición de clase. Por ejemplo:

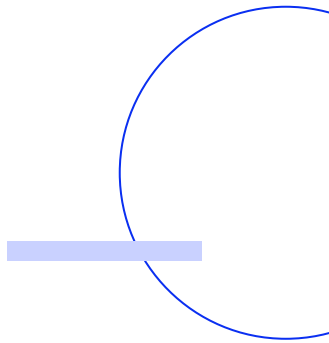
```
public sealed class D
{
    // Aquí los miembros de la clase.
}
```

Una clase sellada no se puede utilizar como clase base. Por esta razón, tampoco puede ser una clase abstracta. Las clases selladas evitan la derivación. Puesto que nunca se pueden utilizar como clase base, algunas optimizaciones en tiempo de ejecución pueden hacer que sea un poco más rápido llamar a miembros de clase sellada.



Un método, indizador, propiedad o evento de una clase derivada que reemplaza a un miembro virtual de la clase base puede declarar ese miembro como sellado. Esto niega el aspecto virtual del miembro para cualquier clase derivada adicional. Esto se logra colocando la palabra clave **sealed** antes de la palabra clave **override** en la declaración del miembro de clase. Por ejemplo:

```
public class D : C
{
    public sealed override void HacerAlgo() {; }
}
```

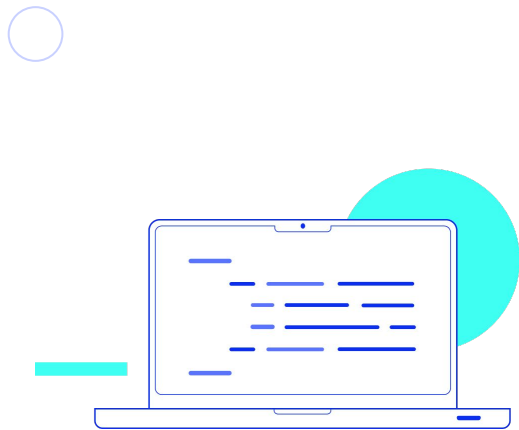


Interfaces

Interfaces

Las interfaces se utilizan para **definir un conjunto de funcionalidades** a través de métodos y atributos que sólo **se definen, pero no se implementan**.

Las interfaces no se utilizan directamente, es decir, no se instancian. Se implementan en otras clases. Una clase puede implementar múltiples interfaces.



Implementación de interfaces

En el siguiente ejemplo vemos cómo una clase **Empleado** puede heredar de la clase **Persona** y a su vez implementa las interfaces **IAdministrar** e **ISubcontratado**:

```
public interface IAdministrar
{
    //Los métodos están definidos pero no implementados
    void Crear();
    void Modificar();
    void Eliminar();
}
```



...

```
public interface ISubcontratado
{
    //Los atributos están definidos pero no implementados
    string Empresa { get; set }
    bool Activo { get; set }
}

public class Empleado:Persona,ISubcontratado,IAdministrar
{
    public string Legajo { get; set }
    public string Empresa { get; set }
    public bool Activo { get; set }

    public void Crear()
    {
        Console.WriteLine("Se ha ejecutado el método crear.");
    }
}
```

...

...

```
public void Modificar()
{
    Console.WriteLine("Se ha ejecutado el método modificar.");
}

public void Eliminar()
{
    Console.WriteLine("Se ha ejecutado el método eliminar.");
}
}
```

**¡Sigamos
trabajando!**