Constructores y Destructores

En simples palabras, El Constructor no es más que un método, un tipo especial de método de una clase que se ejecuta cuando se crea su objeto (clase).

"Un constructor es un método de clase que se ejecuta automáticamente cada vez que se crea el objeto de la clase o cuando se inicializa la clase".

Un Constructor es un tipo especial de método de una clase que se invoca automáticamente cuando se crea una instancia de clase. El Constructor se usa para la inicialización de un objeto y para la asignación de memoria de la clase. El Constructor se usa para inicializar el valor de la clase de los campos privados cada vez que se crea una instancia o un objeto de la clase. El Constructor puede estar sobrecargado.

Cuando no creamos el constructor para la clase, el compilador crea automáticamente un constructor predeterminado para la clase. El nombre del constructor siempre es el mismo del nombre de la clase.

Sintaxis General de un Constructor

```
[Modificador de Acceso] NombreClase([Parámetros])
{
}
```

Tipos de Constructor

- Default Constructor
- Constructor Parametrizado
- Static Constructor
- Private Constructor

Tipos de Constructores

Constructor Default

Los constructores que no han definido ningún parámetro se llaman constructores por defecto. Inicializa el mismo valor de cada instancia de clase.

```
using System;

nombrespace EjemploConstructor

{
```

```
    public class Cliente
    {
        public string ClienteNombre;
        public string ClienteDireccion;
        //Default Constructor
        public Cliente()
        {
            ClienteNombre = "Ángel Arias";
            ClienteDireccion = "Vigo";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Cliente emp = new Cliente();
            Console.WriteLine("Nombre de Cliente :" + emp.ClienteNombr
e);
            Console.WriteLine("Direccion de Cliente :" + emp.ClienteDi
reccion);
            Console.ReadLine();
        }
    }
}
```

Constructor Parametrizado

Un constructor que tiene al menos un parámetro se llama Constructor parametrizado. Usando este tipo de constructor podemos inicializar cada instancia de la clase a diferentes valores.

```
using System;

nombrespace EjemploConstructor
```

```
{
    public class Cliente
    {
        public string ClienteNombre;
        public string ClienteDireccion;
        //Parameterized Constructor
        public Cliente(string nombre, string Direccion)
        {
            ClienteNombre = nombre;
            ClienteDireccion = Direccion;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Cliente emp = new Cliente("Ángel Arias", "Vigo");
            Console.WriteLine("Nombre de Cliente :" + emp.ClienteNombr
e);
            Console.WriteLine("Direccion de Cliente :" + emp.ClienteDi
reccion);
            Console.ReadLine();
        }
    }
}
```

Constructor Static

El constructor estático se utiliza para inicializar cualquier tipo de datos estáticos de la clase o para realizar una acción que debe realizarse solo una vez. El Constructor estático se llama automáticamente antes de la primera instancia de la clase o hace referencia a cualquier dato estático. Se llama solo una vez para cualquier cantidad de clases. Se crea una instancia.

```
using System;
nombrespace EjemploConstructor
{
    public class Cliente
    {
        public static readonly long Baseline;
        public string ClienteNombre;
        public string ClienteDireccion;
        //Static Constructor
        static Cliente()
            {
                Baseline = DateTime.Now.Ticks;
                Console.WriteLine("Static constructor executes first")
;
            }
            //Default Constructor
        public Cliente()
        {
            ClienteNombre = "Ángel Arias";
            ClienteDireccion = "Vigo";
            Console.WriteLine("Se ejecuta después del Constructor
static");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
```

```
            Cliente emp = new Cliente();

            Console.WriteLine("Nombre de Cliente :" + emp.ClienteNombr
e);

            Console.WriteLine("Direccion de Cliente :" + emp.ClienteDi
reccion);

            Console.ReadLine();

        }

    }

}
```

Puntos clave del constructor Static:

- Solo se puede crear un constructor estático en la clase.
- No coge ningún parámetro porque es llamado automáticamente por el CLR o por el modificador de acceso.
- Se llama automáticamente antes de la primera instancia de la clase creada.
- No podemos llamar al constructor estático directamente.

Cuándo usar el constructor estático

El constructor estático es muy útil cuando se crean clases contenedoras para código no administrado. También puede usarlo cuando la clase está usando un archivo de registro y el constructor se usa para escribir entradas.

Constructor privado

El constructor privado es un constructor especial que generalmente se usa en las clases que contienen solo miembros estáticos. Si la clase solo tiene constructores privados y ningún constructor público, entonces no es posible crear una instancia de la clase. Básicamente, el constructor privado impide crear una instancia de la clase. Si desea crear una instancia de la clase que tiene un constructor privado, entonces necesita crear un constructor público junto con un constructor privado.

```
using System;

nombrespace EjemploConstructor

{

    public class Cliente

    {

        private Cliente()

        {}
```

```
        public static int clienteActual;

        public static int IncreamentarCliente()

        {

            return ++clienteActual;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Cliente.clienteActual = 50;

            Console.WriteLine("El Cliente Actual
es:" + Cliente.clienteActual);

            Console.WriteLine("Se ha agregado el
Cliente " + Cliente.IncreamentarCliente());

            Console.ReadLine();

        }

    };

}
```

Características:

- El Constructor no es más que un método especial que inicializa la clase o su tarea para inicializar el objeto de su clase.
- Su nombre debe ser el mismo que el nombre de la clase
- Este es un método especial ya que los constructores no tienen tipos de devolución, ni siquiera están vacíos
- Un constructor no puede devolver un valor porque no tienen un tipo de devolución.
- Un constructor no se puede heredar, aunque una clase derivada puede llamar al constructor de la clase base.
- Una clase tiene al menos un constructor, también conocido como el constructor predeterminado (un constructor sin parámetros)
- Tiene que escribir explícitamente un constructor por defecto mientras sobrecarga los constructores.

- Los múltiples constructores de una clase declarada con un conjunto diferente de parámetros se conoce como sobrecarga de constructor.
- Un constructor puede llamar a otro constructor usando this ().

Destructores

El .NET Framework tiene un mecanismo integrado llamado Garbage Collection (Recolección de Basura) para desasignar la memoria ocupada por los objetos no utilizados. El destructor implementa las instrucciones que se ejecutarán durante el proceso de recolección de basura. Un destructor es una función con el mismo nombre que el nombre de la clase pero que comienza con el carácter ~.

Ejemplo:

```
class Clase
{
public Clase()
{
// constructor
}
~Clase()
{
// Destructor
}
}
```

Recuerde que un destructor no puede tener ningún modificador como privado, público, etc. Si declaramos un destructor con un modificador, el compilador mostrará un error. El destructor también vendrá en una sola forma, sin ningún argumentoo. No existe ningún destructor parametrizado en C #.

Los destructores se invocan automáticamente y no se pueden invocar explícitamente. Un objeto se vuelve elegible para la recolección de basura, cuando ya no está siendo utilizado por la parte activa del programa. La ejecución del destructor puede ocurrir en cualquier momento después de que la instancia u objeto sea elegible para la destrucción.

En C # todas las clases se derivan implícitamente del objeto de la clase base. La clase de objeto contiene un método especial, Finalize (), que todas las clases pueden anular. El mecanismo de recolección de basura en .NET llamará a este método antes que la recolección de basura de los objetos de esta clase. Recuerde que cuando proporcionamos un destructor en una clase, durante el tiempo de compilación, el compilador genera automáticamente el método Finalize (). Eso significa que un destructor y el método Finalize () anulado no pueden coexistir en una clase. El siguiente código generará un error de compilación debido al programa anterior.

```
class Clase
{
~ Clase ()
{
}
protected override void Finalize()
{
```

```
}
}
```

FIN