UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

INTRODUCCIÓN A LA PROGRAMACIÓN C#

3 – MODULO 3: CONSTRUCTORES Y DESTRUCTORES

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

UNIDAD: 3 – Modulo 3

PRESENTACIÓN: En esta unidad se ampliara el concepto de constructores, sus variaciones y usos. Destructores, concepto y usos.

OBJETIVOS

Que los participantes logren: Comprender los conceptos, su implementación en el lenguaje y como usarlos.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

TEMARIO

Constructores

Constructores predeterminados

Un constructor que no toma ningún parámetro se denomina constructor predeterminado. Los constructores predeterminados se invocan cada vez que se crea una instancia de un objeto de una clase o struct mediante el operador new y no se proporciona ningún argumento a new.

```
public class Persona
{
    public int Edad;
    public string Nombre;
}
```

Ejercicio:

Instanciar a la clase Persona, que valores van a tener sus atributos Nombre y Edad ?
Justificar.

Constructores de instancia

Los constructores de instancias se usan para crear e inicializar cualquier variable de miembro de instancia cuando se usa la expresión new para crear un objeto de una clase.

```
class Coordenada
{
    public int x, y;

    // constructor predeterminado
    public Coordenada()
    {
        x = 0;
        y = 0;
    }
}
```

Constructores privados

Un constructor private es un caso especial de constructor de instancia. Se utiliza generalmente en clases que contienen sólo miembros estáticos.

```
    public class Contador
    {
        public static int ValorActual;

        private Contador()
        {

        }

        public static int Incrementar()
        {
            return ++ValorActual;
        }
    }
```

Un uso práctico es la implementación del patrón *Singleton*

```
public class Foo
{

  private Foo (){}

  private Foo FooInstance {get;set;}

  public static Foo GetFooInstance ()
  {
    if(FooInstance == null){
      FooInstance = new Foo();
    }

    return FooInstance;
  }

}
```

Constructores estáticos

Un constructor estático se utiliza para inicializar cualquier dato estático o realizar una acción determinada que solo debe realizarse una vez. Es llamado automáticamente antes de crear la primera instancia o de hacer referencia a cualquier miembro estático.

```
class SimpleClass
{
    // Static variable that must be initialized at run time.
    static readonly long baseline;

    // Static constructor is called at most one time, before any
    // instance constructor is invoked or member is accessed.
    static SimpleClass()
    {
        baseline = DateTime.Now.Ticks; 1Tick = 100 nanosegundos
    }
}
```

- Un constructor estático no permite modificadores de acceso ni tiene parámetros.
- Se le llama automáticamente para inicializar la clase antes de crear la primera instancia o de hacer referencia a cualquier miembro estático.
- El constructor estático no puede ser llamado directamente.
- El usuario no puede controlar cuando se ejecuta el constructor estático en el programa.

Su uso se aplica cuando se necesita hacer una inicialización de un campo estático con un valor que depende de la ejecución.

Ejercicio:

Modelar una clase viaje que represente un viaje de un remisero. La clase viaje debe proveer un método Resumen donde se informe:
- Cuando se comenzó a trabajar
- Cuantos viajes se hicieron
- Tiempo de trabajo

Pregunta:

Cuál es la diferencia entre ***readonly*** *y* ***const*** *? Justificar.*

Encadenamiento de Constructores

Con el objetivo de reutilizar código como premisa, se puede hacer lo siguiente:

```
public Coordenada(int x, int y)
{
    this.x = x;
    this.y = y;
}

//Encadenamiento de constructores
public Coordenada(int x) : this(x, 0)
{
}
```

DESTRUCTORES

Concepto

Los destructores se utilizan para eliminar instancias de clases.

Características

- Los destructores no se pueden definir en structs. Sólo se utilizan con clases.
- Una clase sólo puede tener un destructor.
- Los destructores no se pueden heredar ni sobrecargar.
- No se puede llamar a los destructores. Se invocan automáticamente.
- Un destructor no permite modificadores de acceso ni tiene parámetros.

```
class Coche
{
   ~Coche()  // destructor
   {
      // Sentencias para liberar recursos van aca...
   }
}
```

El destructor llama implícitamente al método Finalize sobre la clase base del objeto. Por lo tanto, el código de destructor anterior se traduce implícitamente al siguiente código:

```
   protected override void Finalize()
   {
      try
      {
         // Cleanup statements...
      }
      finally
      {
         base.Finalize();
      }
   }
```

Por lo tanto, se llaman a los destructores de todas la cadena de la herencia.

Código Administrado (o Manejado) vs Código Nativo

Se conoce como código manejado a aquel que se ejecuta dentro del contexto de la máquina virtual de CLR del framework. Mientras que código nativo es el que se ejecuta directamente sobre la CPU.

Destructor vs Dispose

Su uso aplica principalmente a la liberación de **recursos no manejados**; pero desde la existencia del mecanismo de liberación de recursos de manera explícita por medio de la interfaz **IDisposable**, su uso queda relegado por este mecanismo.

Si estamos manejando conexiones a servicios externos, manejadores de archivos o conexiones de base de datos, siempre tenemos que implementar IDisposable.

Recursos No Manejados

Si es parte del framework netcore: es un recurso manejado.

Si es un DLL nativa de la plataforma o cualquier otra cosa que haya usado los servicios cross plataforma (o P/Invoke) entonces es un recurso que no está administrado, y somos responsables de limpiarlo.

Ejemplo:

```
public void Dispose()
{
    Dispose(true);
    // Le aviso al GC que no lo finalice
    GC.SuppressFinalize(this);
}
protected virtual void Dispose(bool disposing)
{
    if (!is_disposed) // Solo si no esta Disposed
    {
        if (disposing)
        {
            Console.WriteLine("Liberando recursos manejados");
            //Ejemplos
            if (this.databaseConnection != null)
            {
                this.databaseConnection.Dispose();
                this.databaseConnection = null;
            }
            if (this.frameBufferImage != null)
            {
                this.frameBufferImage.Dispose();
                this.frameBufferImage = null;
            }
        }
        Console.WriteLine("Liberando recursos no manejados");
        //Ejemplo
       Win32.DestroyHandle(this.CursorFileBitmapIconServiceHandle);
    }
    this.is_disposed = true;
}
```

Tips para el uso de destructores:

- No declarar destructores vacíos. Implican perdida de performance al encolar llamadas sin sentido.
- No se puede tener control de las llamadas al destructor ya que lo hace el garbage collector.
- Nunca se debe llamar explícitamente al método Collect, esto es atributo exclusive del GC.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

REFERENCIAS

https://msdn.microsoft.com/es-ar/library/ace5hbzh(v=vs.120).aspx
https://msdn.microsoft.com/es-ar/library/k6sa6h87(v=vs.120).aspx
https://msdn.microsoft.com/es-ar/library/k9x6w0hc(v=vs.120).aspx
https://msdn.microsoft.com/es-ar/library/kcfb85a6(v=vs.120).aspx
https://msdn.microsoft.com/es-ar/library/66x5fx1b(v=vs.120).aspx
https://docs.microsoft.com/es-es/dotnet/standard/garbage-collection/implementing-dispose

Fecha	Versión	Observaciones
10/11/2018	1.0	Versión Original