CLASE

Una clase es una construcción que permite crear tipos personalizados propios mediante la agrupación de variables de otros tipos, métodos y eventos. Una clase es como un plano. Define los datos y el comportamiento de un tipo. Si la clase no se declara como estática, el código de cliente puede utilizarla mediante la creación de objetos o instancias que se asignan a una variable. La variable permanece en memoria hasta que todas las referencias a ella están fuera del ámbito. Si la clase se declara como estática, solo existe una copia en memoria y el código de cliente solo puede tener acceso a ella a través de la propia clase y no de una variable de instancia.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EjemploAutomovil
{

    class Automovil
    {

        public Automovil(string marca, string modelo, string color, string chasis)
        {
            this.Marca = marca;
            this.Modelo = modelo;
            this.Color = color;
            this.Chasis = chasis;
        }

        public double Velocidad
        {
            get
            {
                return this.velocidad;
            }
        }


        protected double velocidad = 0;
        public string Marca;
        public string Modelo;
        public string Color;
        public string Chasis;
        private string ac;

    public string Ac
        {
           get
           {
            return ac;
           }
```

```
            set
            {
             ac = value;
            }
        }

        public void Acelerar(double cantidad)
        {
            // Aquí se le dice al motor que aumente las revoluciones pertinentes, y...
            Console.WriteLine("Incrementando la velocidad en {0} km/h", cantidad);
            this.velocidad += cantidad;
        }
        public void Girar(double cantidad)
        {
            // Aquí iría el código para girar
            Console.WriteLine("Girando el Automovil {0} grados", cantidad);
        }


        public void Frenar(double cantidad)
        {
            // Aquí se le dice a los frenos que actúen, y...
            Console.WriteLine("Reduciendo la velocidad en {0} km/h", cantidad);
            this.velocidad -= cantidad;
        }

    }

    class ProgramaAutomovil
    {
        static void Main(string[] args)
        {
            Automovil MiAutomovil = new Automovil("Peugeot", "306", "Azul", "1546876");
            Console.WriteLine("Los datos de mi Automovil son:");
            Console.WriteLine("Marca: {0}", MiAutomovil.Marca);
            Console.WriteLine("Modelo: {0}", MiAutomovil.Modelo);
            Console.WriteLine("Color: {0}", MiAutomovil.Color);
            Console.WriteLine("Número de Chasis: {0}", MiAutomovil.Chasis);
            MiAutomovil.Acelerar(100);
            Console.WriteLine("La velocidad actual es de {0} km/h", MiAutomovil.Velocidad);
            MiAutomovil.Frenar(75);
            Console.WriteLine("La velocidad actual es de {0} km/h", MiAutomovil.Velocidad);
            MiAutomovil.Girar(45);
            MiAutomovil.Ac = "SI";
            Console.WriteLine("AC: {0}",MiAutomovil.Ac);

            Console.ReadKey();

        }
    }
}
```

this (Referencia de C#)

La palabra clave this hace referencia a la instancia actual de la clase y también se utiliza como modificador del primer parámetro de un método de extensión.

```
namespace Programa
{
    class Program
    {

        public class Panda
        {
            public string Nombre { get; set; }
            public Panda Pareja { get; set; }

            public void Relacionarse(Panda companiero)
            {
                Pareja = companiero;
                companiero.Pareja = this;
            }
        }

        public static void Main(string[] args)
        {
            Panda p1 = new Panda { Nombre = "Pandi" };
            Panda p2 = new Panda { Nombre = "Lamy" };
            p1.Relacionarse(p2);
            Console.WriteLine(p1.Nombre);
            Console.WriteLine(p1.Pareja.Nombre);
            Console.ReadLine();

        }
    }
}
```

Cuándo Usar la Referencia this

Calificación de Miembros de Instancia Homónimos

Se refiere a los miembros de instancia, en particular campos que tienen el mimo nombre que variables locales en métodos. Veamos:

```
public AdministradorTI (string nombre, string email)

{

    this.nombre = nombre;
    this.email = email;

}
```

En las sentencias del constructor **AdministradorTI** se califica los campos nombre y email por medio de la referencia this. Si en en lo de lo anterior hubiérmos hecho esto:

```
public AdministradorTI (string nombre, string email)
{
    nombre = nombre;
    email = email;
}
```

Las dos sentencias corresponden con lo que se llama **asignacion reduntate**.

Contexto Estático

En C# el uso de la referencia this no está permitido. Si intentamos hacerlo:

```
public static void VerInstanciasCreadas()

{
    Console.WriteLine ("Número de pandas creados: " + numeroPandasCreados);
    this.Relacionarse(new Panda ("Ross"));

}
```

error CS0026: Keyword `this' is not valid in a static property, static method, or static field initializer

OTRA CLASE

```
namespace Programa
{
    class Program
    {
        public class Agenda
        {
            float version = 1.0f; // variable privada
            // no puede salir de la clase
            // solo puede ser usada
            // dentro de la clase
            public string Nombre; // poniendo "public" delante
            // de cualquier funcion
            // o variable, habilitamos
            // su uso al invocar la clase
            private int Sexo;
```

```
            public void DameTuNombre()
            {
                if (Version() == 1.0f)
                {
                    Console.WriteLine("Cual es tu nombre?");
                    Nombre = Console.ReadLine();
                }
            }
            // podemos sobrecargar una función
            // en otras palabras, crear la misma
            // función con diferentes argumentos
            public void DameTuNombre(string nombre)
            {
                Nombre = nombre;
            }
            public string DameTu(string algo)
            {
                Console.WriteLine("Escribe tu {0,0:s}", algo);
                Nombre = Console.ReadLine();
                return Nombre;
            }
            private float Version()
            {
                return version;
            }
        }


        public static void Main(string[] args)
        {
            int numero = 3;
            string cadena = "mi cadena";

            Agenda A = new Agenda();

            A.Nombre = "Pedro";

            A.DameTuNombre();
            A.DameTuNombre("Pepito");
            cadena = A.DameTu("el nombre de tu abuela");

            Console.WriteLine("Hola {0}! como estas?", cadena);

            Console.ReadLine();

        }
    }
}
```

CLASE ABSTRACTA

Una clase abstracta puede definir variables de instancia, mientras que una interface no puede hacerlo. Una clase abstracta puede implementar métodos, mientras que una interface no puede hacerlo.
Si una clase hereda de una clase abstracta, ésta no puede heredar de otra clase; si una clase implementa una interface, tiene la libertad de heredar de otra clase e implementar muchas otras interfaces. Una clase abstracta debe ser utilizada cuando algunos detalles en la implementación, como las variables y definiciones de métodos, son comunes para todas las clases en una jerarquía.

```
public abstract class Producto
  {
      protected string nombre;
      protected double precio;
      protected double costo;

      public Producto(string nombre, double precio, double costo)
      {
          this.nombre = nombre;
          this.precio = precio;
          this.costo = costo;
      }

      public abstract string imprimirDatos();
  }

  public class Libro : Producto
  {
      public Libro(string titulo, double precio, double costo)
        : base(titulo, precio, costo)
      {

      }

      public override string imprimirDatos()
      {
          return "Libro: " + nombre + ", Precio: " + precio;
      }
  }

  public class DVD : Producto
  {
      public DVD(string titulo, double precio, double costo)
          : base(titulo, precio, costo)
      {

      }
```

```
        public override string imprimirDatos()
        {
            return "DVD: " + nombre + " Precio: " + precio;
        }

    }


    static void Main(string[] args)
    {

        Libro miLibro = new Libro("Biblioteca del programador", 54.95, 39.95);
        DVD miDVD = new DVD("Curso multimedia de Java", 29.95, 19.95);

        Console.WriteLine("Los datos de mis productos.");
        Console.WriteLine(miLibro.imprimirDatos());
        Console.WriteLine(miDVD.imprimirDatos());


        Console.WriteLine("presione <enter> para terminar.");
        Console.ReadLine();
    }
```

INTERFACE

Una interface define un conjunto de métodos abstractos; algunas interfaces incluyen también declaraciones de constantes.

Una interface debe ser utilizada para definir un comportamiento que debe ser implementado por clases de diferentes jerarquías.

```
    interface Iuno
    {
        void AccionUno();
    }

    class Implementa : Iuno
    {
        public void AccionUno()
        {
            Console.WriteLine("Acción uno...");
        }
    }

    static void Main(string[] args)
    {
        Implementa I = new Implementa();
                I.AccionUno();
        Console.ReadLine();
          }
```