

**Instituto Tecnológico Nacional de México**

**Instituto tecnológico del Sur de Nayarit**

**Programación Orientada a Objetos**

Tema 4: Polimorfismo y Herencia

Interfaces en C#

Cinthia Anahí Mata Bravo

**Docente**

Gema Guadalupe Álvarez Cervantes

**Estudiante**

**Semestre:** 2

**Matricula:** 191140057

**Fecha de Entrega:** 20-marzo-2020

**Que son Interfaces:** Describen un grupo de funciones relacionadas, que pueden pertenecer a cualquier clase o estructura. Básicamente se trata de representar un contrato que debe cumplir cualquier clase que implemente la interfaz.

**Implementación:** NO poseen implementaciones.

**Herencias:** Herencia Múltiple

**Modificadores de Acceso:** Automáticamente todos sus miembros son públicos, como se observa en la imagen

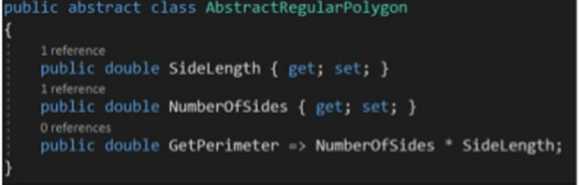
Campos
Propiedades
Contructores
Destructores
Métodos
Eventos
Indices
Propiedades
Métodos
Eventos
Indices
Comparación: Miem...

**Miembros Válidos:** Propiedades, Métodos, Eventos, índices.

**¿Que son las clases abstractas?** Son mecanismos que obligan la herencia No se pueden instanciar, es decir, no se puede crear objetos de ellas Se utilizan solamente para heredar de ellas (Forzar u obligar la herencia).

Se antepone la palabra “abstract” al nombre de la clase. Se puede sobrescribir los miembros, métodos y funciones de una clase abstracta, e incluso extender la clase que herede de ella con las particularidades que consideremos oportunas.

**Implementación:** Tiene implementaciones



**Herencias:** Herencia Simple

**Modificadores de Acceso:** Puede tener modificadores de acceso

Campos
Propiedades
Contructores
Destructores
Métodos
Eventos
Indices
Propiedades
Métodos
Eventos
Indices
Comparación: Miem...

**Miembros Válidos:** Campos, Propiedades Constructores, Destructores Métodos, Eventos, Índices.

En el paradigma orientado a objetos son una colección de acciones o propiedades que definen una clase en este caso nuestra interface será IAve la cual contendrá un par de métodos que todas las aves realizan Volar() y Comer().

public interface IAve

{

void Volar();

void Comer();

}

Ahora crearemos una clase la cual implementara la interface anteriormente realizada.

public class AvePropiedades:IAve

{

public int Patas=2;

public int Alas=2;

public int Cola=1;

public string Nombre{ get; private set;}

public AvePropiedades(string NAve)

{

this.Nombre = NAve;

}

public void Volar()

{

Console.WriteLine ("Volar");

}

public void Comer()

{

Console.WriteLine ("Comer");

}

}

Como vemos hemos implementado ambos métodos de nuestra interface(Volar();, Comer();) en caso de no implementarlos se generaría un error debido a que estas propiedades o métodos deben de ser obligatorios para poder implementar la interface en nuestra clase.

Ahora crearemos otra clase en la cual implementaremos nuestra clase AvePropiedades la cual contiene un constructor el cual nos inicializa la variable Nombre.

public class Ave:AvePropiedades

{

public Ave(string Nombre):base(Nombre)

{

}

}

Como podemos notar nuestro constructor de la clase Ave nos pide como parámetro un string el cual inicializara nuestra clase padre o base AvePropiedades.

Enseguida implementaremos nuestra clase Ave en un codigo simple.

class MainClass

{

public static void Main (string[] args)

{

//Instanciamos dos clases del tipo ave pasando de parámetro el nombre del ave

Ave canario = new Ave ("Canario");

Ave cuervo = new Ave ("Cuervo");

//Creamos una instancia de nuestra clase principal para llamar el método imprimir ()

MainClass main = new MainClass ();

main.imprimir (canario);

main.imprimir (cuervo);

Console.ReadKey ();

}

///

/// Imprimir the specified ave.

/// Este metodo imprimira las propiedades de una clase ave.

///

/// Ave.

public void imprimir(Ave ave)

{

//Imprimimos el nombre del ave, el número de patas y el número de colas

Console.WriteLine ("Soy un {0} tengo {1} Pata(s), {2} Ala(s) y {3} Cola(s)", ave.Nombre, ave.Patas, ave.Alas, ave.Cola);

//Llamamos a nuestros dos metodos de la interface Comer() y Volar()

Console.WriteLine ("Puedo ");

ave.Comer ();

Console.WriteLine ("y tambien puedo ");

ave.Volar ();

}

}