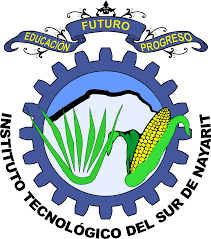
**Materia:** Programación Orientada a Objetos

**Nombre:** Miramontes Luis Uriel

**Ejercicios de la**

**Unidad II y III**

**Maestra:** Ing. Cinthia Anahí Mata Bravo



***27/02/2020***

**Investigación de “Interface” (Interfaz) C#**

Una interfaz contiene definiciones para un grupo de funcionalidades relacionadas que una clase no abstracta o una estructura deben implementar. Una interfaz puede definir staticmétodos, que deben tener una implementación. Una interfaz puede proporcionar una implementación predeterminada para cualquiera o todos sus miembros de instancia declarados. Una interfaz no puede declarar datos de instancia como campos, propiedades implementadas automáticamente o eventos similares a propiedades.

Mediante el uso de interfaces, puede, por ejemplo, incluir el comportamiento de múltiples fuentes en una clase. Esa capacidad es importante en C # porque el lenguaje no admite la herencia múltiple de clases. Además, debe usar una interfaz si desea simular la herencia de estructuras, porque en realidad no pueden heredar de otra estructura o clase.

Usted define una interfaz utilizando la palabra clave de interfaz como se muestra en el siguiente ejemplo.

C#

interface IEquatable<T>

{

bool Equals(T obj);

}

El nombre de una interfaz debe ser un nombre de identificador de C # válido. Por convención, los nombres de las interfaces comienzan con mayúscula.

Cualquier clase o estructura que implemente la interfaz IEquatable <T> debe contener una definición para un método Equals que coincida con la firma que especifica la interfaz. Como resultado, puede contar con una clase que se implemente IEquatable<T>para contener un Equals método con el que una instancia de la clase puede determinar si es igual a otra instancia de la misma clase.

La definición de IEquatable<T>no proporciona una implementación para Equals. Una clase o estructura puede implementar múltiples interfaces, pero una clase solo puede heredar de una sola clase.

Las interfaces pueden contener métodos de instancia, propiedades, eventos, indexadores o cualquier combinación de esos cuatro tipos de miembros. Las interfaces pueden contener constructores estáticos, campos, constantes u operadores. Una interfaz no puede contener campos de instancias, constructores de instancias o finalizadores. Los miembros de la interfaz son públicos por defecto.

Para implementar un miembro de interfaz, el miembro correspondiente de la clase de implementación debe ser público, no estático y tener el mismo nombre y firma que el miembro de interfaz.

Cuando una clase o estructura implementa una interfaz, la clase o estructura debe proporcionar una implementación para todos los miembros que la interfaz declara, pero no proporciona una implementación predeterminada. Sin embargo, si una clase base implementa una interfaz, cualquier clase derivada de la clase base hereda esa implementación.

El siguiente ejemplo muestra una implementación de la interfaz IEquatable <T>. La clase implementadora Carro, debe proporcionar una implementación del método Equals.

C#

public class Carro: IEquatable<Carro>

{

public string HechoEn {get; set;}

public string Modelo { get; set; }

public string Año { get; set; }

// Implementation of IEquatable<T> interface

public bool Equals(Carro carro)

{

return (this.HechoEn, this.Modelo, this.Año) ==

(carro.HechoEn, carro.Modelo, carro.Año);

}

}

Las propiedades e indexadores de una clase pueden definir accesores adicionales para una propiedad o indexador que se define en una interfaz. Por ejemplo, una interfaz puede declarar una propiedad que tiene un descriptor de acceso get . La clase que implementa la interfaz puede declarar la misma propiedad con a gety set accessor. Sin embargo, si la propiedad o el indexador usa una implementación explícita, los accesores deben coincidir. Las interfaces pueden heredar de una o más interfaces. La interfaz derivada hereda los miembros de sus interfaces base. Una clase que implementa una interfaz derivada debe implementar todos los miembros en la interfaz derivada, incluidos todos los miembros de las interfaces base de la interfaz derivada. Esa clase se puede convertir implícitamente a la interfaz derivada o cualquiera de sus interfaces base. Una clase puede incluir una interfaz varias veces a través de clases base que hereda o a través de interfaces que otras interfaces heredan. Sin embargo, la clase puede proporcionar una implementación de una interfaz solo una vez y solo si la clase declara la interfaz como parte de la definición de la clase (class ClassName : InterfaceName) Si la interfaz se hereda porque usted heredó una clase base que implementa la interfaz, la clase base proporciona la implementación de los miembros de la interfaz. Sin embargo, la clase derivada puede volver a implementar cualquier miembro de la interfaz virtual en lugar de utilizar la implementación heredada. Cuando las interfaces declaran una implementación predeterminada de un método, cualquier clase que implemente esa interfaz hereda esa implementación. Las implementaciones definidas en las interfaces son virtuales y la clase implementadora puede anular esa implementación.

Una clase base también puede implementar miembros de la interfaz mediante el uso de miembros virtuales. En ese caso, una clase derivada puede cambiar el comportamiento de la interfaz anulando los miembros virtuales.

**Resumen personal**

Una interfaz en si es técnicamente como una clase base abstracta con más clases ya sean abstractas, padres e hijas. Cualquier clase o estructura que implemente la interfaz debe implementar todos sus miembros. Opcionalmente, una interfaz puede definir implementaciones predeterminadas para algunos o todos sus miembros.

Una interfaz no puede ser instanciada (que no tiene variables sino atributos) directamente. Sus miembros son implementados por cualquier clase o estructura que implemente la interfaz. Mientras que una clase o estructura puede implementar múltiples interfaces. Una clase puede heredar una clase base y también implementar una o más interfaces.

Según la programación orientada a objetos, la interfaz de una clase es todo lo que podemos hacer con ella. A efectos prácticos: todos los métodos, propiedades y variables públicas (aunque no debería haber nunca variables públicas, debemos usar propiedades en su lugar) de la clase conforman su interfaz.

Tanto las clases abstractas como las interfaces son mecanismos que obligan la

herencia. No se pueden instanciar, es decir, no se puede crear objetos de ellas.

Además, las interfaces son mecanismos para que puedan interactuar varios objetos no relacionados entre sí, son protocolos o “contratos” que obligan la herencia que contienen las declaraciones de los métodos, pero no su implementación. Al igual que las clases abstractas, son plantillas de comportamiento que deben ser implementados por otras clases.

La interfaz de una clase define el comportamiento de dicha clase, ya que define que podemos y que no podemos hacer con objetos de dicha clase.

En general siempre que tengas, o preveas que puedes tener más de una clase para hacer lo mismo: es mejor usar interfaces.

No toda clase debe implementar una interfaz obligatoriamente, muchas clases internas no lo implementan, pero en el caso de las clases públicas (que son visibles desde el exterior).Además, pensar en la interfaz antes que, en la clase en sí, es pensar en lo que debe hacerse en lugar de pensar en cómo debe hacerse. Usar interfaces permite cambiar una clase por otra que implemente la misma interfaz y poder integrar la nueva clase de forma mucho más fácil (sólo se debe modificar donde instanciamos los objetos, pero el resto de código queda igual).

**Programa “InterfaceMaquinaExpendedora”**

Este programa simula una máquina expendedora de refrescos, solamente que esta solo dice los precios de los productos dentro de ella. El objetivo del programa en sí es cuando al momento de que el usuario escoja algún producto de la máquina expendedora, le diga el producto seleccionado y el precio por medio de un mensaje.

Para esto se necesita crear un “interface” donde esta solamente se crea una propiedad que en este caso se llama “tipoDeProducto” que únicamente lo que hace es heredarla a otras clases para así poder sobrescribir en ella.

Ya terminado lo anterior se crea una clase padre llamada “Producto” y una clase hija llamada “Bebidas” donde en la clase padre se necesita usar el atributo “refresco” para poder seleccionar los tipos de refrescos de la máquina que vienen siendo “Lata 350ml” y “Botella 600ml” (para esto se necesita la propiedad del “interface”), mientras que en la clase hija se necesita usar los atributos “cocacola” y “drPepper” que vienen siendo los refrescos dentro de la máquina para así conocer los precios de estas mismas (para esto igualmente se necesita la propiedad del “interface”).

Ya completado lo antes mencionado se necesita crear un diseño llamativo con imágenes de los productos de Coca-Cola y DrPepper, después se debe mostrar los tipos de refrescos para que el usuario pueda seleccionarlos y poner algún botón para saber los precios del refresco seleccionado.

(Este programa se ubica en la carpeta Unidad4 😊)